UNIVERSITÀ
DI TRENTO

Department of Information Engineering and Computer Science

Bachelor's Degree in
Ingegneria Informatica, Elettronica e delle Telecomunicazioni

Final dissertation

Engineering of a data acquisition tool for drone-based hyperspectral camera images quantitative radiometric corrections

Supervisor

Lorenzo Bruzzone

Student

Simone Tollardo

Academic year 2021/2022

Acknowledgements

I begin by expressing the thanks to my family and my parents who, in their way, always allowed me to study and follow my passions and my dreams, things that unfortunately not everyone can do. A big thank should also go to Sara that always helped me get through tough times and stressful periods, especially during the Covid-19 pandemic. I am really grateful to all of those with whom I have had the pleasure to study, work and enjoy these years of university, starting from Tommaso, Lisa, Elia, Ruben, Thomas, Filippo, Mattia, Nicola, Alex to all the colleagues and friends that I found in E-Agle FSAE team and in university rooms. I would like to thank all the professors and university staff that taught with passion and dedication feeding my curiosity and my desire to learn. I would finally like to express my deep gratitude to the thesis supervisor Professor Lorenzo Bruzzone, director of the RSLab, for making this wonderful work possible and to the co-supervisor Massimo Santoni, who helped me in an admirably way with passionate support, enthusiasm and a big big willingness.

Trento, September 03, 2022 *Simone Tollardo*

Contents

Abstract

The **growing use of hyperspectral cameras** has extended the scope of these tools to purposes that need to cover **large areas with low altitudes**, such as remote sensing in general but also **precision agriculture** and **environmental monitoring**. Nowadays this is more and more done with the help of **autonomous Unmanned Aerial Vehicles**. Since the most of the UAV-based hyperspectral sensors are usually bi-dimensional (line-scanners), they require physical movement in order to proceed to scan the next line. This process clearly takes a defined time that depends firstly on the sensor and obviously also on the aircraft speed.

For this reason, covering large areas is prone to **time-variations of the incoming irradiance**, typically because of the **movement of clouds**. This could results in big inaccuracies, especially if the application requires quantitative radiometric data.

Many solutions to this problem exploit the correlation in time, in wavelength or spatial domains of the images to replace the shadowed pixels with others whose values are calculated by complex algorithms. Other more elaborate methods use in addition geometrical and radiative transfer models to improve the accuracy of the **estimated values**. The main problem of these methods is the fact that the replacement values are estimated and do not necessarily reflect real values, resulting in poor performances whenever the purpose is to obtain precise quantitative data.
Only few methods utilize the **incoming irradiance**, usually measured by **expensive and closed-source instrumentation**, to compensate for shadows. These relatively recent approaches clearly require more studies and investigations but in general they showed decent results.

Therefore, the aim of this thesis, is the development of an **affordable open-source solution**, from the hardware to the frontend, that allows to capture the necessary information to **compensate with high precision** for the discussed unwanted artifacts **using real and simultaneously recorded data**.

1 Introduction

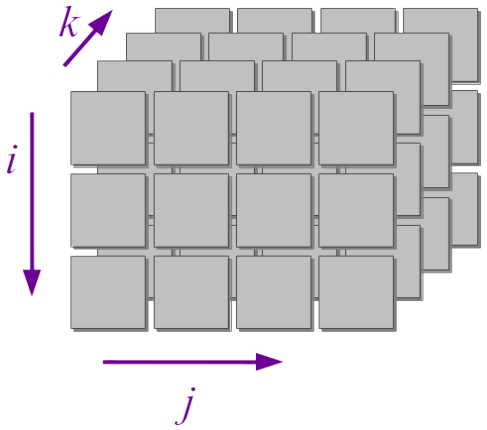
1.1 Hyperspectral imaging

Taking a picture means **collecting the information about the amplitude of the electromagnetic (EM) field** for specific portions of the EM spectrum, usually the visible one, which starts from a wavelength of 380nm and ends at about 750nm.

In digital imaging, these information are typically stored in ***n-D arrays*** (also called ***Datacubes***) whose elements are called ***pixels***. A color image, for example, can be represented with an array of three dimensions: X, Y and color (e.g. Red, Green and Blue), while for a monochromatic image only two dimensions are needed: X and Y. Obviously, colors are just specific frequencies of the EM spectrum or, in reality, specific portions of it. If the acquisition system can not discriminate **more than one EM spectrum portion**, the resulting array will be *bi*-Dimensional, and therefore, a simple matrix, otherwise, the frequency (or wavelength) variable must be taken into account and the array will be at least *three*-Dimensional (there is also the possibility to add to the game other variables, such as time, for example).

$$\begin{array}{c|cccc} & 1 & 2 & \dots & n \\ 1 & a_{11} & a_{12} & \dots & a_{1n} \\ 2 & a_{21} & a_{22} & \dots & a_{2n} \\ 3 & a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \dots & a_{mn} \end{array}$$

(a) *bi*-Dimensional array of n columns and m rows



(b) *three*-Dimensional array of j columns, i rows and k "slices"

Figure 1.1: ***bi*-dimensional** and ***three*-dimensional** arrays.

With *Multispectral Imaging* we typically refer to the capture of images with more than 3 and less than 15 spectral bands, while with *Hyperspectral Imaging* we typically refer to the capture of images with **more than 15 contiguous bands**, ideally forming a continuous spectrum.

Multispectral and hyperspectral cameras, which are cameras that can acquire respectively multispectral and hyperspectral images, are widely used for *Remote Sensing* purposes. However nowadays, multispectral and hyperspectral cameras are starting to be widespread also in numerous other applications fields such as precision agriculture, environmental monitoring, pharmaceutical production, artworks, biochemistry but also waste sorting, forensic sciences, archaeology, or entomology.

1.1.1 Resolution of multispectral and hyperspectral images

The resolution of an image is defined as the total pixel count of the image, thus, considering for example an image of 1000 pixels along X direction and 1000 pixels along Y direction has resolution of: $1000 * 1000P = 1 * 10^6P = 1MP$. Since in multispectral and hyperspectral images the frequency (or

wavelength) variable appears, we call ***Spatial Resolution*** the resolution of the acquisition system in the spatial domain (typically the XY plane), and ***Spectral Resolution*** the smallest separation in wavelength that the system can distinguish ($\Delta\lambda = \frac{\lambda}{R}$, where R is the so called *Resolving Power*).

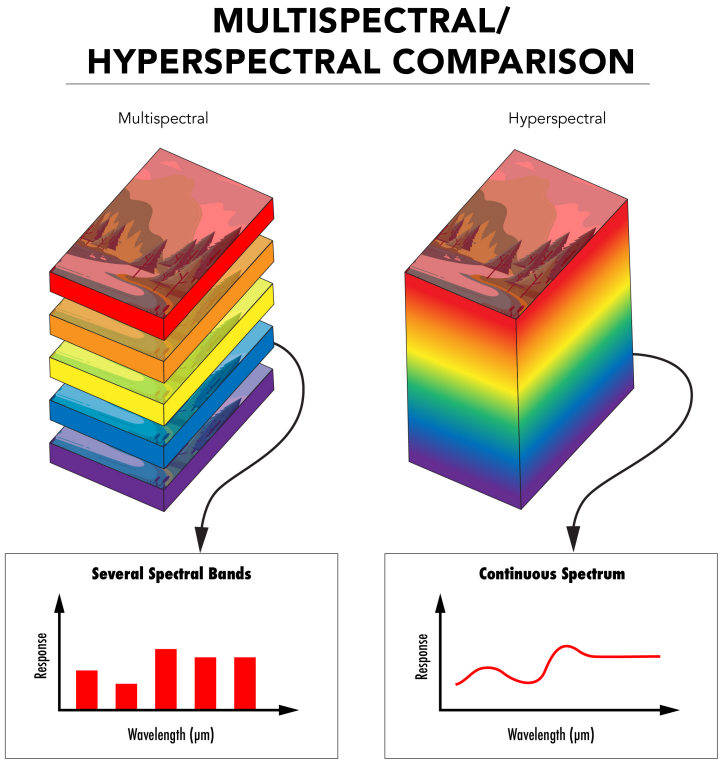


Figure 1.2: Multispectral imaging vs hyperspectral imaging.

1.1.2 Overview of the RSLab's Hyperspectral camera

Let us give some fast overview of the hyperspectral camera considered for this thesis, the Mjolnir HySpex V-1240.

The Mjolnir HySpex V-1240 is 4kg complete Unmanned Aerial Vehicle (UAV) hyperspectral acquisition unit. It contains the hyperspectral camera itself, the data acquisition unit and the navigation system.

The specifications of the hyperspectral camera are as follows:

- **Spectral range**: 400-1000nm
- **Spatial resolution**: 1240x1
- **Spectral channels**: 400
- **Spectral resolution**: 3nm
- **Bit resolution**: 12 bit
- **FOV**: 20°
- **Dynamic range**: 4400

Scanning technique

As it can be guessed from the specifications, **the camera collects datas per lines**. This technique is called ***Spatial Scanning***, in which the sensor is a 2-D (x, λ) sensor, and the final image is obtained by physically moving this so called ***Push Broom Scanner*** towards the spatial direction perpendicular

to the sensor spatial scanning direction (y). This can be done with the help of satellites, planes, UAVs and so on.

The focus of this thesis will be mainly on the UAV-based data collection method.

The spectral resolution of the camera is dictated by the optics, in fact the number of real distinguishable channels is 200. In practise, during the acquisition, the camera sensor is oversampling every 3nm interval with a $\frac{1000nm-400nm}{400channels} = 1.5nm$ interval (double sampling frequency).

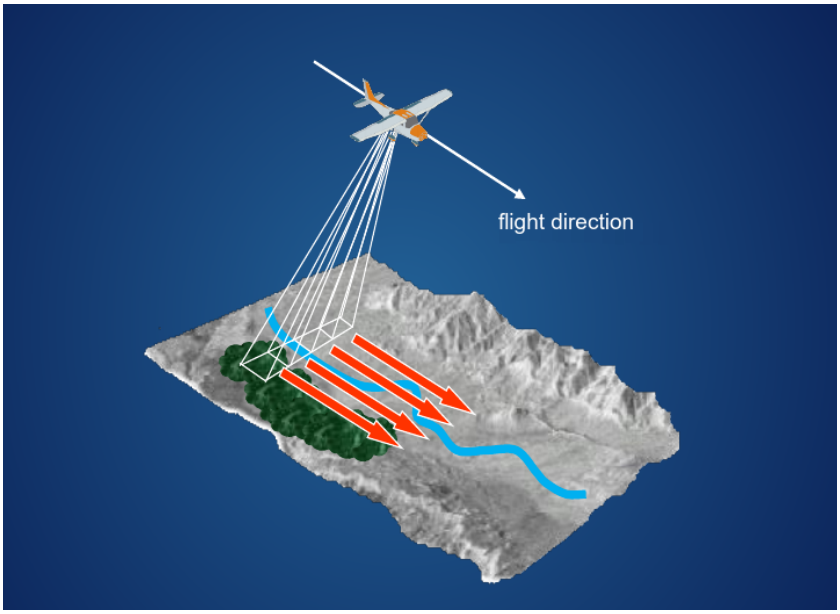


Figure 1.3: Push Broom Scanner

Data processing steps

The processing steps made with acquired data taken during the flight mission are the following:

1. **Radiometric calibration of the RAW data**
2. **Geo-coding**
3. **Othorectification**
4. **Atmospheric Correction**

The radiometric calibration first converts the RAW DN into **at-sensor *radiance*** values, then *Smile effect* and *Keystone effect* are corrected and a second data calibration based on the environmental temperature measured during the flight is carried out.

Geo-coding principally means geo-referencing the acquired data based on the acquisition position (usually given by a Global Positioning System (GPS) receiver); in this way **every pixel of the image corresponds to a precise location**. In our case, for a better precision, a differential GPS is used: one GPS is installed on the UAV and one is positioned in mission's base.

During the orthorectification process *Tilt effect* and *Relief effect* are removed so that the image is *planimetrically correct* and thus it has a constant scale wherein **features are represented in their true position**. This allows for accurate direct measurement of distances, angles, and areas. The input required for the orthorectification process is the Digital Surface Model (DSM), or Digital Elevation Model (DEM), this is usually provided by LiDAR scannings.

The last step, the *Atmospheric Correction*, aims to **eliminate atmospheric and illumination effects** such as haze, clouds etc. The main actor of this step is the widely used software *ATCOR-4*.

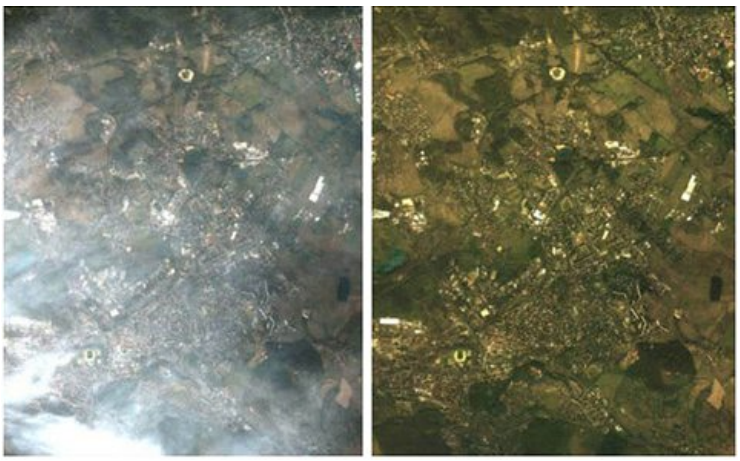


Figure 1.4: Image before (left) and after (right) ATCOR correction.

1.2 Problem statement

The camera sensor measures the radiation reflected or emitted by the interested surface. It's trivial to understand that **the measured radiance depends on the amount of light that hit the surface**, and since the used scanning technique requires **non negligible time to scan all the surface**, if this value is time-dependent, it could lead to non ideal results: different areas of the image could for example result brighter than other even if in reality they were not.

How can we know if a **darker area on an image effectively corresponds to a less reflective surface** and not to a moving cloud shadow, a simple object shadow or just the sunset?

The answer is to measure the *Spectral Directional Reflectance* of a surface at any time instant $R_{\Omega,t}$:

$$R_{\Omega} = \frac{L^{\mathrm{r}}_{\mathrm{e},\Omega,t}}{L^{\mathrm{i}}_{\mathrm{e},\Omega,t}},$$

where:

- $L^{\mathrm{r}}_{\mathrm{e},\Omega,t}$ is the *Spectral Radiance* **Reflected** by the surface at any time instant;
- $L^{\mathrm{i}}_{\mathrm{e},\Omega,t}$ is the *Spectral Radiance* **Received** by the surface at any time instant.

So, in order to obtain the reflectance of the interested surface, for every infinitesimal part of the surface we should measure the radiance in two "points": a point before the light hit the infinitesimal surface (and so it is the received radiance), and a point after the light has hit the infinitesimal surface (and so it is the reflected radiance).

The measure of every "second point" (in reality the measure is done for every Ground Sample Interval (GSI)) is the role of the hyperspectral camera while the measure of every "first point" would be clearly impractical (the interested surface can not be filled with sensors!). A possible solution would be measuring the integration in the spatial domain of the radiance received by entire surface or by multiple finite surface portions.

The *Radiance* of a surface at any time instant is defined as follows:

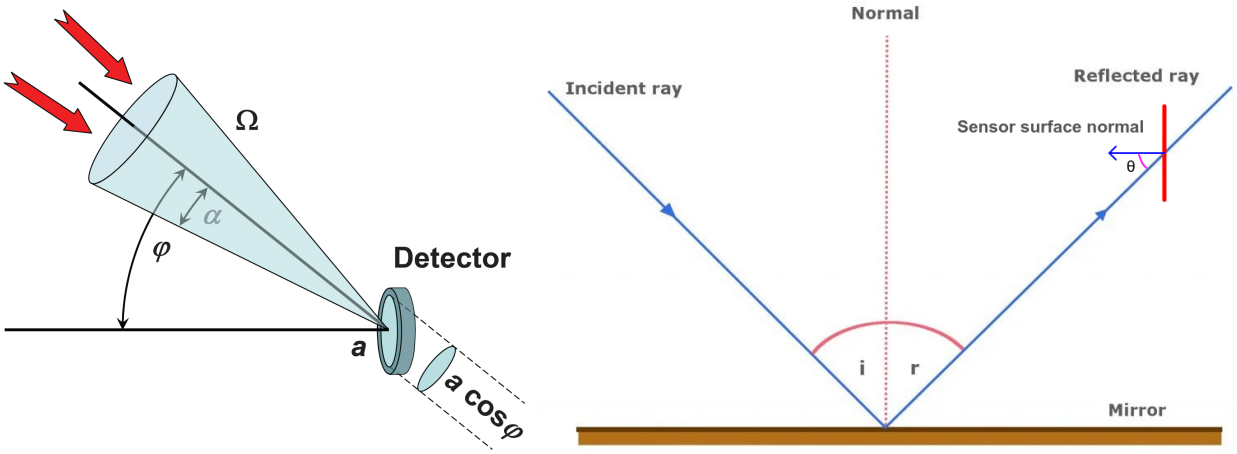
$$L_{\mathrm{e},\Omega,\mathrm{t}} = \frac{\partial^2 \Phi_{\mathrm{e,t}}}{\partial \Omega_{\mathrm{t}}\, \partial (A_{\mathrm{t}} \cos \theta_{\mathrm{t}})},$$

where:

- ∂ is the partial derivative symbol;

- $\Phi_{e,t}$ is the *Radiant Flux* emitted, reflected, transmitted or received;
- Ω is the *solid angle*;
- $A\cos\theta$ is the *projected area*.
- t is time

Now, considering that we are measuring the light reflected from the ground, and that the only light source is the sun, we can state that the projected area depends only on the sensor surface and on the *viewing angle* θ_t. Thus, we can derive $A_t \cos\theta_t = A\cos\theta_t$, where θ represents the angle between the normal to the sensor surface and the reflected beam. According to the law of reflection, since the angle of reflection θ^r is always equal to the angle of incidence θ^i, considering a locally plane surface and a negligible variation in time of the angle between the camera sensor and the ground, we can state that the angle θ_t depends only on the actual *Solar Zenith Angle*.



(a) Graphical visualization of projected area $a\cos\phi$ and solid angle Ω.

(b) Graphical visualization of the angle θ.

Figure 1.5: Graphical visualization of the considered geometry

For what concern the solid angle Ω_t, we can say that it is the Field Of View (FOV) of the camera sensor, thus it only depends on the camera sensor's distance from the ground, on the area of the sensor itself and on the camera lens. For these reasons, the solid angle Ω_t can be assumed not time-dependent: $\Omega_t = \Omega$ or at least easily computable.

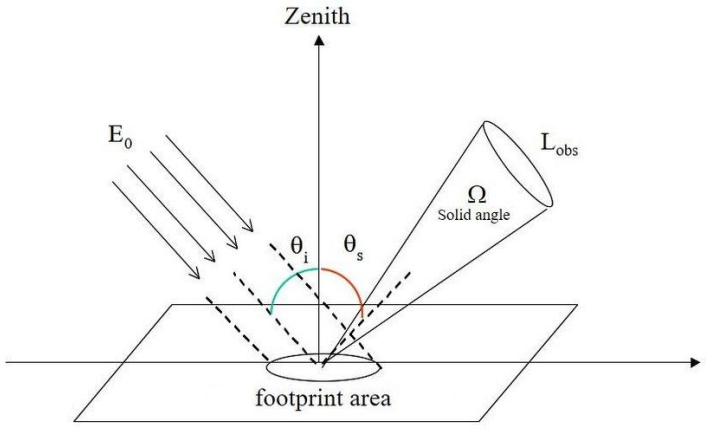


Figure 1.6: Graphical visualization of the solid angle of the of the sensor.

Since the solar zenith angle can be easily obtained with simple algorithms knowing time and position of the camera, in order to keep the measured radiance non time-dependent but only surface-dependent, the only thing we need to know is the exact value of $\Phi_{\mathrm{e,t}}$ at every time instant.

However, since we are interested in small portions of the spectrum and not in the total effect of the radiation of all wavelengths we need to introduce first the *Spectral Reflectance* ***in wavelength*** of a surface and then the *Spectral Radiance* ***in wavelength*** of a surface at any time instant:

$$R_{\Omega,\lambda} = \frac{L^{\mathrm{r}}_{\mathrm{e},\Omega,t,\lambda}}{L^{\mathrm{i}}_{\mathrm{e},\Omega,t,\lambda}},$$

where:

- $L^{\mathrm{r}}_{\mathrm{e},\Omega,t,\lambda}$ is the *Spectral Radiance in wavelength* **reflected** by the surface at any time instant
- $L^{\mathrm{i}}_{\mathrm{e},\Omega,t,\lambda}$ is the *Spectral Radiance in wavelength* **received** by the surface at any time instant.

The Spectral Radiance in Wavelength at any time instant is defined as follow:

$$L_{\mathrm{e},\Omega,t,\lambda} = \frac{\partial L_{\mathrm{e},\Omega,t}}{\partial \lambda},$$

where λ is the wavelength.

Applying the same considerations stated before, we can extract the following equation:

$$L_{\mathrm{e},\Omega,t,\lambda} = \frac{\partial L_{\mathrm{e},\Omega,t}}{\partial \lambda} = \frac{\partial^2 \Phi_{\mathrm{e,t},\lambda}}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})},$$

where: $\Phi_{\mathrm{e,t},\lambda} = \dfrac{\partial \Phi_{\mathrm{e,t}}}{\partial \lambda}$ is the *Radiant Flux* ***in wavelength***.

Since the camera can not measure infinitesimal portions of the spectrum but instead it can integrates small portions of this, we need to integrate in the wavelength domain the radiance for finite intervals (1.5nm in this case).

Firstly, substituting radiant flux in wavelength we can obtain:

$$L_{\mathrm{e},\Omega,t,\lambda} = \frac{\partial L_{\mathrm{e},\Omega,t}}{\partial \lambda} = \frac{\partial^3 \Phi_{\mathrm{e,t}}}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})\,\partial\lambda}.$$

Then, integrating in wavelength domain and knowing that Ω as well as A and $\cos\theta$ are wavelength-independent:

$$\Delta L_{\mathrm{e},\Omega,t} = L_{\mathrm{e},\Omega,t}(\lambda_2) - L_{\mathrm{e},\Omega,t}(\lambda_1) = \int_{L_{\mathrm{e},\Omega,t}(\lambda_1)}^{L_{\mathrm{e},\Omega,t}(\lambda_2)} L_{\mathrm{e},\Omega,t,\lambda}\,\partial\lambda = \int_{\lambda_1}^{\lambda_2} \partial L_{\mathrm{e},\Omega,t} = \int_{\lambda_1}^{\lambda_2} \frac{\partial^2 \Phi_{\mathrm{e,t}}}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})}$$

$$\int_{\lambda_1}^{\lambda_2} \frac{\partial^2 \Phi_{\mathrm{e,t}}}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})} = \frac{\partial^2}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})} \int_{\lambda_1}^{\lambda_2} \Phi_{\mathrm{e,t}} = \frac{\partial^2}{\partial\Omega\,\partial(A\cos\theta_{\mathrm{t}})} \int_{\lambda_1}^{\lambda_2} \Phi_{\mathrm{e,t},\lambda}\,\partial\lambda.$$

Finally, knowing that the integral of $\Phi_{\mathrm{e,t},\lambda}$ is

$$\Delta\Phi_{\mathrm{e,t}} = \Phi_{\mathrm{e,t}}(\lambda_2) - \Phi_{\mathrm{e,t}}(\lambda_1) = \int_{\Phi_{\mathrm{e}}(\lambda_1)}^{\Phi_{\mathrm{e}}(\lambda_2)} \partial\Phi_{\mathrm{e,t}} = \int_{\lambda_1}^{\lambda_2} \partial\Phi_{\mathrm{e},\lambda}\,\partial\lambda,$$

we can state:

$$\Delta L_{\mathrm{e},\Omega,t} = L_{\mathrm{e},\Omega,t}(\lambda_2) - L_{\mathrm{e},\Omega,t}(\lambda_1) = \frac{\partial^2 \Delta \Phi_{\mathrm{e,t}}}{\partial \Omega \, \partial (A \cos \theta_{\mathrm{t}})}.$$

This means that, if we could have a way to **retrieve the received radiance at any time instant**, and ideally for every smallest surface possible, we could apply corrections to the hyperspectral camera images and **calculate the reflectance of the surface**, solving the initial problem.

1.3 Background

While in literature it's easy to find shadow detection and shadow removal solutions, finding one that replaces shadowed areas with **radiometric accurate data** is not that simple. Moreover, as well as being mainly focused on satellite images, almost all of them work with optical clouds data (or estimations of them) to retrieve the reflectance value of the shadowed surface.

For what concern the state-of-the-art of shadow detection for UAV-based hyperspectral imaging, the method proposed by Liu et al., utilizing LiDAR data to do the segmentation and extract homogenous regions, seems very promising, with result accuracy up to 99.5%!

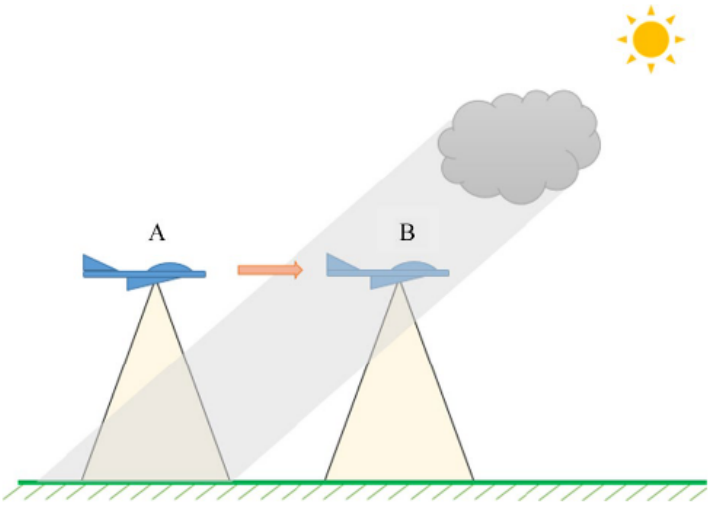


Figure 1.7: Visual representation of cloud shadow.

Regarding shadow removal, as stated in [10], in recent years, numerous cloud/shadow removal methods have been proposed; these methods can be split in three categories:

- ***Self-Complementation based***: full use of the information from cloud/shadow-free regions; the contaminated information of cloud/shadow regions is supplemented by propagating the geometrical flow from the cloud/shadow-free regions;
- ***Multispectral-Complementation based***: utilize the strong correlation in different spectral bands to reconstruct the image;
- ***Multitemporal-Complementation based***: utilize prior images to complement the contaminated information.

An example of the first category is the paper [8]. The proposed method makes use of reference images to detect clouds and cloud shadows. Then, the novel Multi-temporal Cloud Removal (MCR) algorithm is used to replace the cloud and cloud shadow contaminated pixels with same pixels of the reference

image in the target image.

However, this method does not fit our purpose for four main reasons:

- the reflectance value of the reference pixel do not necessarily match the actual value of the surface;
- multi-temporal reference images may be not available;
- the method is meant for high altitude images, not feasible with UAVs;
- the method is mainly focused on multispectral images.

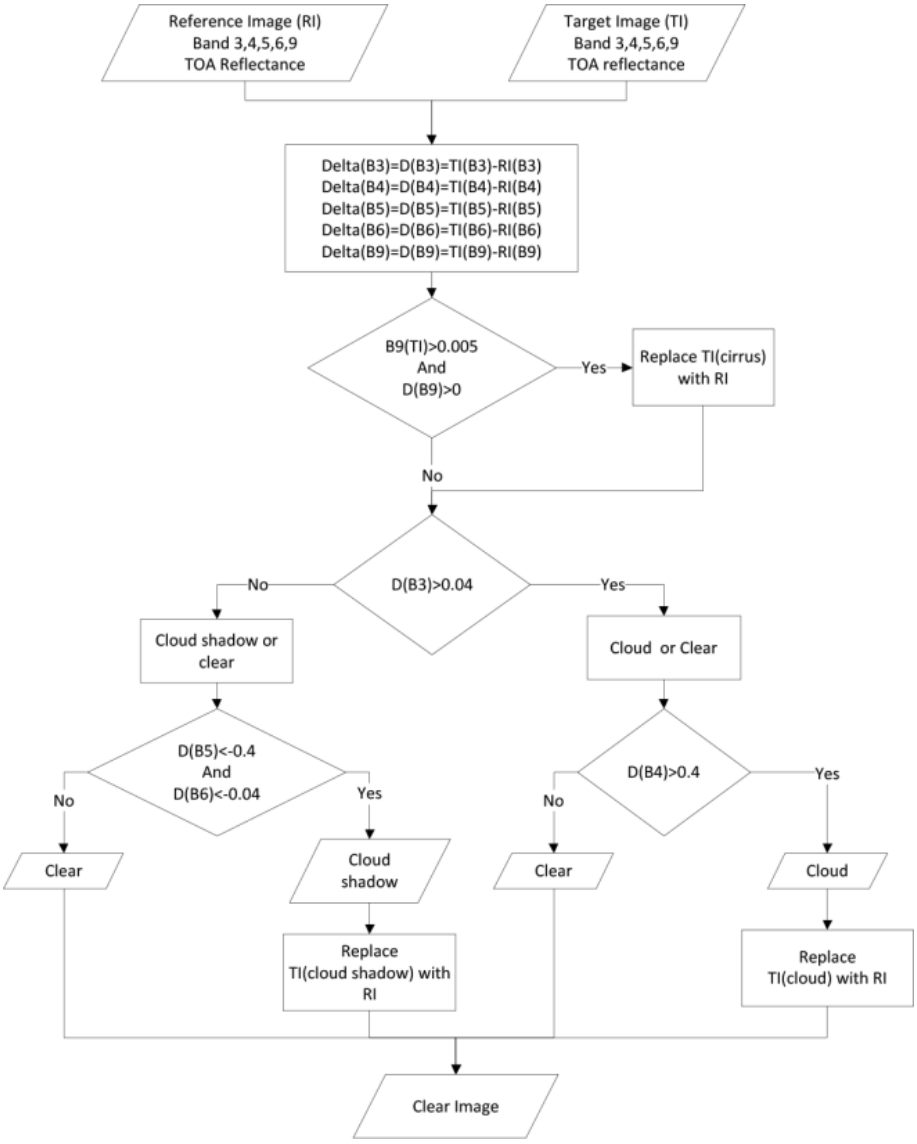


Figure 1.8: Flowchart of the MCR algorithm.

Another paper analyzed, [35], brings a mixture of both the self-complementation and the multispectral-complementation approaches: the authors propose a method to retreive the reflectance of the shadowed image pixels exploiting the attenuation factor for the solar irradiance. The method makes use of already known algorithms that employ geometric model of the position of the sun, of the satellite, and of the cloud along with spectral properties of shadows to detect cloud shadowed pixels.

To correct pixels in the areas shadowed by clouds, a *radiative transfer model* is employed. Considering the model, incident solar irradiance on the ground has two components which are ***direct solar irra-***

diance and ***diffused solar irradiance***. The direct solar irradiance gets attenuated by clouds, which casts a shadow on the ground, while diffused irradiance does not. Thus, a shadowed area receives partial direct solar irradiance and full diffused solar irradiance. Therefore, a shadow pixel retains some information of the ground. This retained information can be refined to retrieve the ground reflectance if the Attenuation Factor for the direct Solar Irradiance (AFSI) at the pixel is known. An AFSI value is a fraction of the direct solar irradiance received by the ground. An AFSI value of zero indicates full shadow, while the value of one indicates no shadow. [35]

Now, this paper brings a new method to retrieve AFSI values with better reliability in terms of radiometric accuracy than conventional methods (e.g. the de-shadow method proposed in the ATCOR package). The limitations of the conventional methods are: the processing of non-shadowed pixels, sometimes resulting in artifacts, the fact that every water pixel should be removed beforehand otherwise over-correction occurs and the low reliability of the AFSI values estimation. The new proposed method derives instead the Attenuation Factor for the direct Solar Irradiance values from the optical properties of an occluding cloud estimating the cloud transmittance. The method requires the cloud and the cloud-shadow to coexist in the input image. Only shadowed pixels are processed.

For the scope of this thesis this method has some major problems, too:

- the solution needs high altitude images (cloud and the cloud-shadow must coexist in the image), not feasible with UAVs
- results show relatively low average correlation coefficient between corrected image and reference image (0.75)

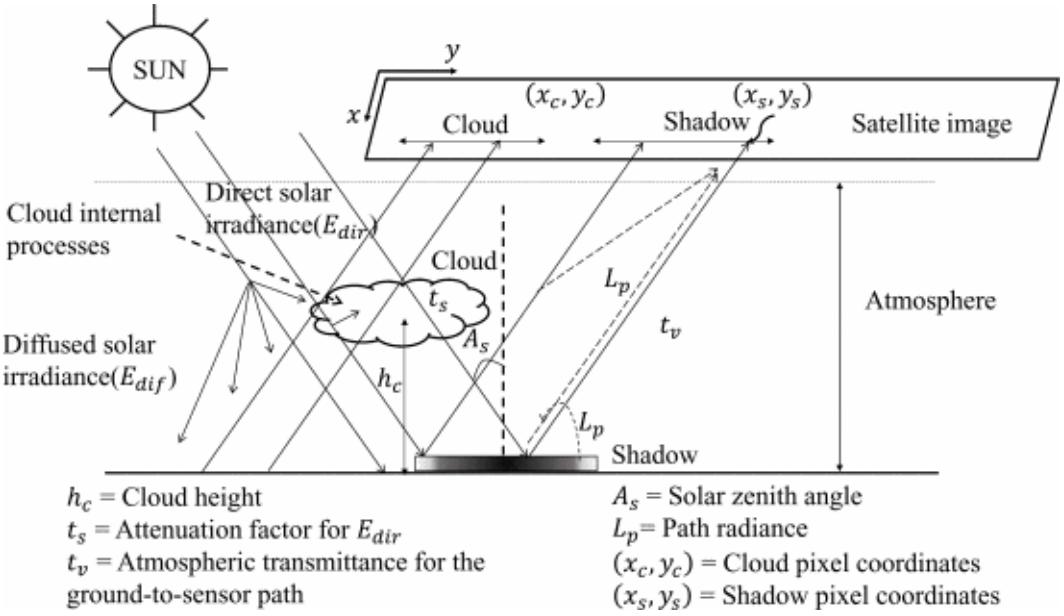


Figure 1.9: Representation of a physical model of cloud shadow formation.

Among all these methods, the *low-rank matrix/tensor completion* based methods (ML), such as [10], [37], [29], which utilize spatial, spectral, and temporal information simultaneously, have been applied to the cloud/shadow removal problem and achieved the state-of-the-art results. However, this last methods requires first of all large amounts of data and high demand of computing power, moreover **all recovered data are anyway estimated** and do not necessarily precisely reflect real quantities.

Solutions that make use of **simultaneously recorded incoming irradiance to compensate for cloud shadow** are [5] and [19]; they make uses of proprietary and very expensive hardware such as **irradiance sensors or *spectrometers*** in combination with ***reference reflectance panels*** to extract the reflectance of the interested surface. The sensor used by first method was only one, embedded in the camera, mounted on UAV. The orientation of the sensor and its details are not clear, the only pro-

vided information is that it can measure irradiance in 5 bands: 465–485nm, 550–570nm, 663–673nm, 712–722nm, 820–860nm. The second method uses instead two spectrometers: one mounted on the UAV and one positioned on the ground. The first one has a range of 350-1000nm while the second one 350-2500nm. In [5], results showed that reference reflectance panels were very important and effective, while irradiance sensors were not always useful and sometimes their data had low correlation with the multispectral ones. Apparently this was mainly caused by the not always stable attitude of the drone during the flight, in fact, small tilt angle of this one, as showed in [19], resulted in drastic changes of the measured radiance of the sensor and for this reason Hakala et al. use two sensors.

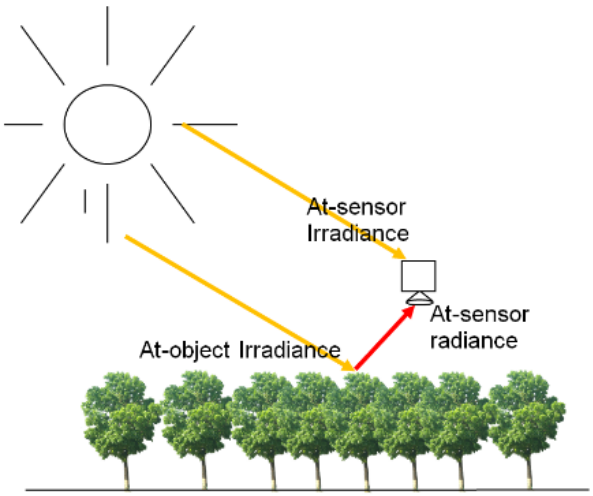


Figure 1.10: Principle of the direct reflectance measurement from a drone.

Given all these considerations, the aim of the solution I am proposing in next chapters is developed to solve multiple typical complications that affect this particular type of radiometric corrections. In any case, the solution is not only intended as a stand-alone solution but instead a technique that could provide accurate real data to already known State-of-the-Art methods and thus enhance quality of results. Moreover these data could not only be used to remove cloud-shadow artifacts but also to improve other numerous types of reflectance corrections (e.g. normal objects' shadows and reflections, low altitude fog and aerosols scattering, unwanted solar irradiance effects etc.).

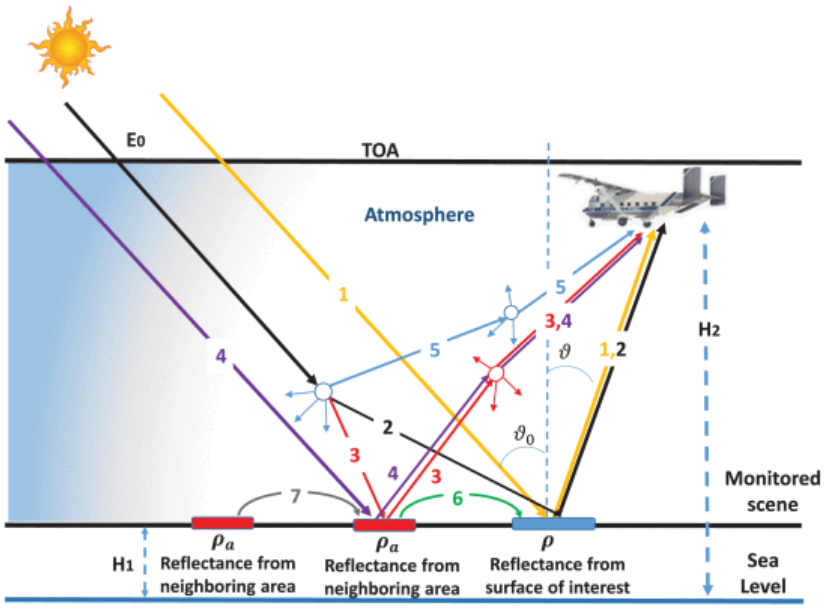


Figure 1.11: Examples of unwanted effects.

2 LAITS Concepts

A way to retrieve our spectral radiance received by the surface, at any time instant, for given wavelength intervals $\Delta L^{\mathrm{i}}_{\mathrm{e},\Omega,t}$, is to have another detector that measures directly the received (by the surface) radiance of the light source, similarly to those proposed by Barker et al. and Hakala et al..

The rules that the solution must follows are nine:

1. The system is intended to be an IoT sensors network capable of running simultaneously a number of theoretically infinite nodes;
2. The system must be able to effectively capture relevant data from a static position but also when fitted on board of the UAV without any modification;
3. In order to have indicative measures the system must cover a large part of the camera spectrum, ideally with at least five bands;
4. Every recorded data must be precisely geo-referenced and time-referenced;
5. The solution must be portable and versatile;
6. The solution must be practical and user-friendly;
7. The solution must be reliable;
8. The solution must be secure;
9. The solution must be affordable.

As a result of these capabilities, first requirements were established:

- The system requires a custom designed PCB that brings all the components needed to develop the features;
- In order to receive, parse and store data, a back-end subsystem is required;
- A front-end interface that allows the user to visualize and download the recorder data is necessary.

From all the requirements a concept was born, *LAITS*: the Light Analysis Iot porTable Spectrometer.

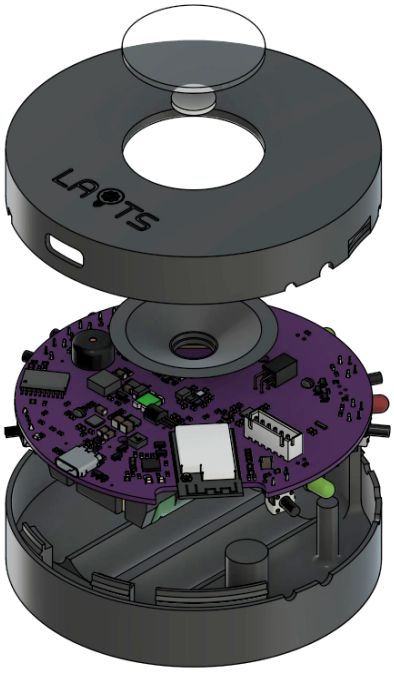


Figure 2.1: Rendering of the LAITS Prototype

3 LAITS Hardware

The design of LAITS started from the hardware that allows the system to work.
Considering market availability and prices, the first part of the design was the choice of the main components such as sensors, MicroController Unit (MCU), Real-Time Clock (RTC) and GPS, USB and Power management.

3.1 Components choice and Schematic

3.1.1 Sensors

Surely the most important parts of the spectrometer are the sensors. Two possibilities open-up: the preferable option consists in a **CMOS 11 bands spectral sensor** by AMS, AS7341, while the second one, much less accurate, exploits three sensors, namely TCS34725FN, BH1750FVI and TSL2561. The spectral response of the AS7341 covers the wavelengths **from about 350nm to 1000nm with 11 bands** while the second solution can cover the same spectrum portion with only 7 bands. The second solution was provided in case of AS7341 runs out-of-stock and because of the availability of old TCS34725 and BH1750 breakout boards in University's Lab. The second solution do not fully comply with rule N°3 of the LAITS concepts. Both solutions are compatible with LAITS Hardware V1.0.x and can also coexist and work together if desired. All these sensors provide an Inter-Integrated Circuit (I2C) full-speed interface. A DS18B20 Dallas 1-Wire temperature sensor is installed on the board for **temperature compensation**.

Figure 3.1: AMS AS7341-DLGM

3.1.2 MCU

The second most important component of the Printed Circuit Board (PCB) is certainly the MCU. Since Wi-Fi connectivity in this application is a must to comply with rule N°1, the very famous ***Espressif ESP32*** MCU serie suits the best. Castellated holes all-in-one modules are available for most of the ESP32 MCUs. The main ESP available series are the following:

- ESP8266 Series
- ESP32 Series
- ESP32-S Series
- ESP32-C Series

Among all these series, the only AIO modules, non-NRND, supporting Bluetooth Low Energy (BLE) and with enough GPIOs available on the market were:

- ESP32-C3-MINI-1U
- ESP32-C3-WROOM-02U
- ESP32-WROOM-32UE
- ESP32-WROVER-IE

Now, considering that ESP32-C3 is single core 160MHz while ESP32 is dual core 240MHz, that ESP32-C3 has only 15 GPIOs available while ESP32 has got 26 (WROOM-32UE) or 24 (WROVER-IE) of them, the ESP32-C3 was discarded from the choice. Since ESP32-WROVER-IE has both a PCB antenna and an *IPEX* connector, its 8MB PSRAM version was chosen as the winner.
NOTE: The default position of the antenna selector jumper connects the IPEX antenna; if the PCB antenna is used the jumper needs to be moved to the other position.

Figure 3.2: Espressif ESP32-WROVER-IE

3.1.3 RTC and GPS

In order to satisfy the rule N°4, if we want to keep the time stored even in absence of internet connection an RTC or a GPS should be mounted. Regarding the geo-referentiation of the data, this can be precisely done only if a GPS is present on the PCB. Here, in the same way as in the Sensors section, two options are available: **placing only the RTC** PCF2127AT or **placing only the GPS** UBLOX SAM-M8Q (placing both would not have any benefits since GPS can retrieve time from satellites). This choice was made because of the much higher price of the GPS module with respect to the RTC and thus the impact on the price of the whole board, resulting in a non compliance with rule N°9 of the concepts. Again, both peripherals offer a full-speed I2C interface.

3.1.4 Micro SD Card

As stated in rule N°7, the system must be **reliable**: for this reason, the board is provided with a *Micro Secure Digital (SD)* Card. In this card all the **data are stored as they are recorded**. In this way, if one of the nodes disconnects from the access point or for energy saving purposes the user chose to not connect it, all the data can be simply accessed reading the content of the card.

3.1.5 USB and Power Management

As stated in rule N°5, the prototype needs to be portable and versatile and that is why it is equipped with two *Li-Ion 18650* batteries. Moreover, in order to comply with rule N°6, the system should be able to recharge batteries without the needs of any other hardware. To do this, the PCB comes with an up-to-date **USB type C** connector, a SILABS CP2102N USB to UART controller capable of handling BC detect, USB 2.0 Full Speed (12Mbps) and Baud rates up to 3Mbaud, and a Maxim Integrated MAX77960B **3A buck-boost charger**.

3.1.6 Auxiliaries

To make LAITS as user-friendly as possible, additional LEDs, buttons and a buzzer were added to notify states and facilitate interaction with the board.

After all the components researches, the next step was the PCB design itself using the open-source EDA-CAD *KiCad 6.0.7*. The first step was the creation of the missing KiCad symbols and footprints (some of the components used were not present on KiCad default library) using the component editor, then the design of the schematic took place.

Considering rule N°6, in order to easily interact with the system, six buttons are been provided: a reset button, a *bootloader mode* selection button, a microSD log enable button, a *provisioning reset* button (see Chapter 4: LAITS Firmware and Chapter 5: LAITS APP), a RGB led enable (show

board status on RGB LED), and a sleep button. The first two buttons are not exposed to the user from version 1.0.3 since there is no need to do so. For the same reasons, the board is provided with a battery charge LED, a RGB led, a microSD status LED, 3.3V status one and UART TX/TX LEDs. In addition, automatic bootloader and reset sequences through DTS and RTS UART flow control signals are implemented in both hardware and CP2102N configuration for easy USB flashing, debugging and monitoring.

Regarding rule N°7 of LAITS concepts, all possible failures have been covered by numerous protections in order to **minimize the possibility of issues**. Examples of them could be ESD protection for every hands-reachable ESD sensitive component, decoupling capacitors for every IC supply pins, overcurrent, overvoltage and reverse polarity protection on main power supply line, predisposition of custom EMI solutions for USB shielding, predisposition of components to mitigate known silicon issues on ICs, use of 1.8V level shifters with 3.3V tolerant lines in order to remove any possible premature components failures (see AMS AS7341 datasheet for details), zener protection on battery voltage sensing lines etc.

Results can be seen on the attachments sections A.1 LAITS-Hardware Schematics and A.3 LAITS-Hardware BOM.

3.2 Layout

Starting from the schematic, the next steps have been: importing the *netlist* and the footprints into the *Layout Editor*, configuring Design Rule Checker (DRC) rules and the board stackup and finally start components placement and traces routing.

The board stackup is a 1.6mm **4 layers** type, with **external copper thickness of 70um** (or $2oz/ft^2$ copper area density) and internal one of 15,2um (or $\frac{1}{2}oz/ft^2$ copper area density).
The reason for the higher-price 70um copper thickness (instead of the classic 35um) lies in the fact that there is the need to lower temperatures when high battery recharge current is flowing. In fact, the MAX77960 can support up to 3A charge current, while the package offers only a maximum of 0.3mm of contact thickness! For this reason, every high-power trace of the MAX77960 was routed through all layers apart from the *inner layer 1* (GND plane) which needed to be as intact and wide as possible. This was done using a lot of small VIAs (mostly 0.2/0.6mm) and *copper pours*.

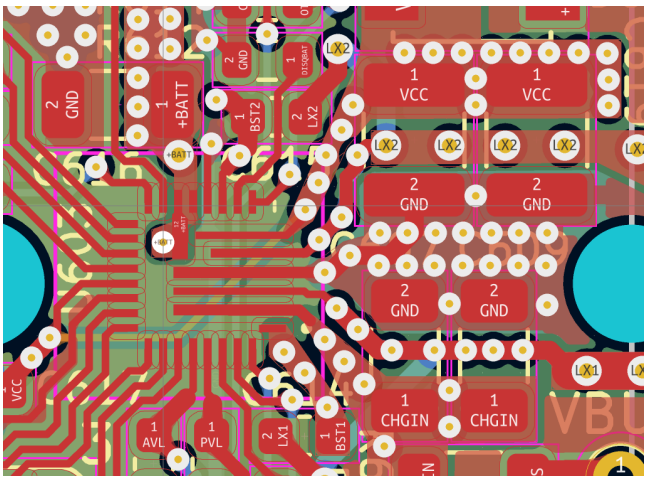
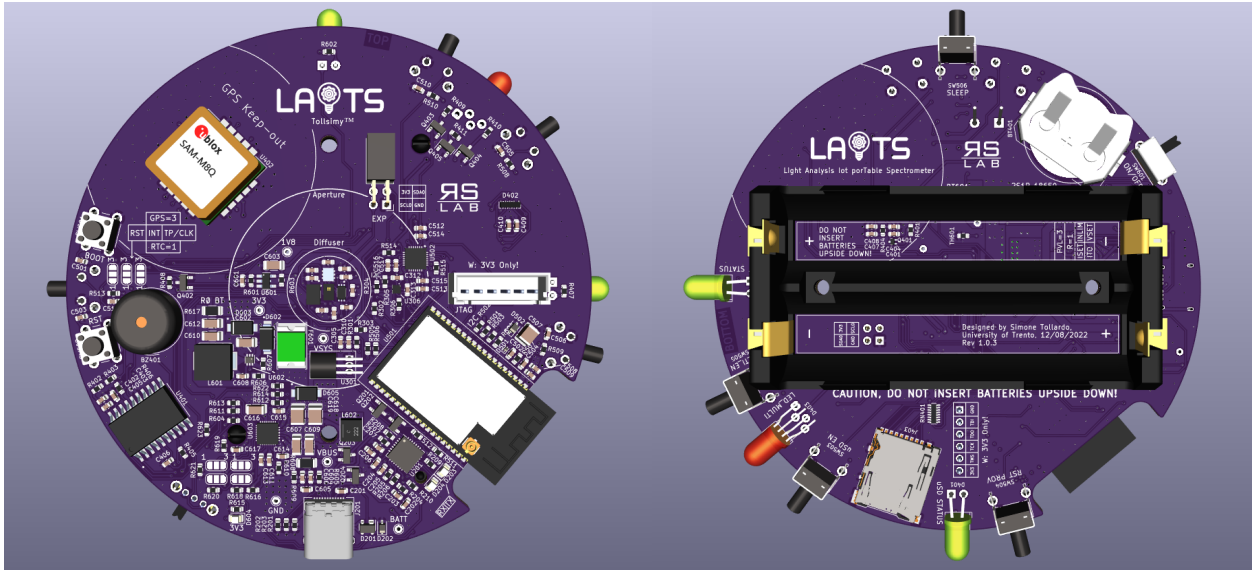


Figure 3.3: MAX77960 IC traces routing

Other layout special needs were the GPS *Keepout Area* and *Ground Plane*, the ESP PCB antenna keepout, both required for a good signal reception (and on the last one also signal transmission), and the USB routing. The former was done with the help of the impedance calculator, based on the JLC7628 stackup, and with the aid of the KiCad differential pair routing tool.

The results and all the layers can be seen in Layout attachments section A.2 LAITS-Hardware Gerber layers.



(a) TOP (b) BOTTOM

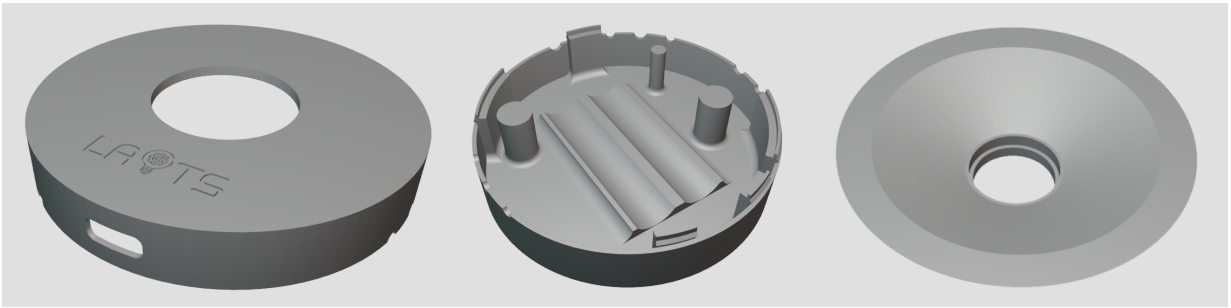
Figure 3.4: Rendering of LAITS Hardware v1.0.3

As for the rule N°5, the total cost of a single board as July 2022, amounts to **less than 150€**.

3.3 Enclosure and optics

Since the light reflected from the ground should not interfere with the measure of incident radiance, LAITS sensors' **FOV need to be less than 180°** but also not too small, otherwise relevant areas may be excluded from the measure. According to the first mentioned reason the material of the enclosure should be less transmissive as possible. Following the AMS AS7341 datasheet [2], on top of the sensor an ***achromatic optical diffuser*** should be placed. The recommended scattering characteristic is quasi-cosine and it is shown in datasheet. A **protective lens** should also be placed on top of the enclosure so that dust can not enter the enclosure and damage sensors.

From these requirements, a **3D printed enclosure** has been designed. The casing is composed of three parts: a bottom body, a top body and the aperture body. Top body and bottom body are ready to be printed with any FDM printer while the aperture body should be printed as precise and smooth as possible. Example top and bottom bodies were printed in a opaque black PLA. A functional aperture body was printed with a small SLA resin printer. Exploded view of all components can be seen in Figure 2.1.



(a) Top body (b) Bottom body (c) Aperture body

Figure 3.5: Rendering of LAITS-Enclosure for LAITS-Hardware v1.0.0

4 LAITS Firmware

This chapter of the thesis, structured in macro blocks, provides all the major information about the program that runs on the LAITS board (on the ESP32 MCU and on the CP2102N IC).

4.1 CP2102N Configuration

Before flashing the ESP the USB to UART controller needs to be configured in order to properly handle USB communication and USB Battery Charger 1.2.

To do so, Silabs provides *Simplicity Studio*. Simplicity Studio is a tool that offers, among other numerous features, the possibility to launch *Xpress Configurator*, from which it is possible to compile and **flash the CP2102N configuration**.

From Xpress Configurator there is the possibility to set the behaviour of the GPIOs of the IC, enable or disable the BC, change the USB PID and VID, change the Power Mode (Self-Powered or Bus-Powered), set the allowed UART Baudrates and a lot more.

After flashing the correct configuration (available on the repository), the ESP32 can be easily programmed trough a normal USB CDC port.

4.2 ESP32

Multiple cross-compilers are available for ESP32 platform: *C/C++* compilers, *MicroPython* bytecode compilers and interpreters and LLVM-based compiler for *Rust*, *TinyGo* and so on.

Multiple frameworks are also available: most famous are the *Arduino Core* and the Espressif official framework: ***ESP-IDF*** (Espressif IoT Development Framework).

4.2.1 Toolchain and ESP-IDF

ESP-IDF is an open source development framework that supports a large number of software components such as a multicore FreeRTOS kernel, peripheral drivers, network stack protocols implementations, security features and a lot more. It supports officially only C and C++ programming languages, however, it is possible to use the resources that it provides with other already mentioned compilers. It is really well documented and easy to use but it also lets the possibility to run complex and extensive tasks without hiding the low-level implementations of the APIs and HALs.

All the firmware that runs on the ESP of the LAITS board is developed using ESP-IDF V4.4.

4.2.2 Drivers

Each of the I2C-based peripherals needs a driver in order to define a sort of **HAL to facilitate program readability and re-usability**. As August 2022 the following peripherals can count a working driver: AS7341, BH1750, TCS34725, DS18B20, MAX77960, PCF2127, SX1503. All drivers are being developed in parallel to the LAITS Firmware which is in fact somehow independent from them (see RTOS Tasks section). Every driver mainly consists of a *component* folder and a *main* folder. The former contains an example project that makes use of the peripheral's driver while the first one contains all the the HAL's source files.

Normally an embedded C/C++ driver is split into *Header* files and *Source* files. Header files typically contain registers definitions, enumerated types and constants, driver structures declarations and HAL functions declarations. Source files typically provide definitions of the declared HAL functions.
All LAITS' drivers follow the same logic for uniformity reasons.

Registers definitions

Register definitions are just preprocessor directives that define a symbol as a numeric constant, a string, etc. First of all every peripheral register's name is mapped with the hexadecimal number that represent its address. Then, if the 8 bit register content represents an entire field nothing else is written, else way, every field is defined with an **offset** and a **lenght**. The lenght represents the number of bits that the field has while the offset represents the number of the first bit that corresponds to that field.

CHG_CNFG_01 (0x17)

Charger configuration 1

BIT	7	6	5	4	3	2	1	0
Field	PQEN	LPM	CHG_RSTRT[1:0]		STAT_EN	FCHGTIME[2:0]		
Reset	0x1	0x0	0x1		0x1	0x1		
Access Type	Write, Read	Write, Read	Write, Read		Write, Read	Write, Read		

Figure 4.1: Example of multiple fields register; e.g. CHG RSTRT: Offset=4, Lenght=2

Enumerated types

In C language, enumerated types are user-defined data types. They are mainly used to assign names to integral constants, so the program is easy to read and to maintain.

Driver Structures

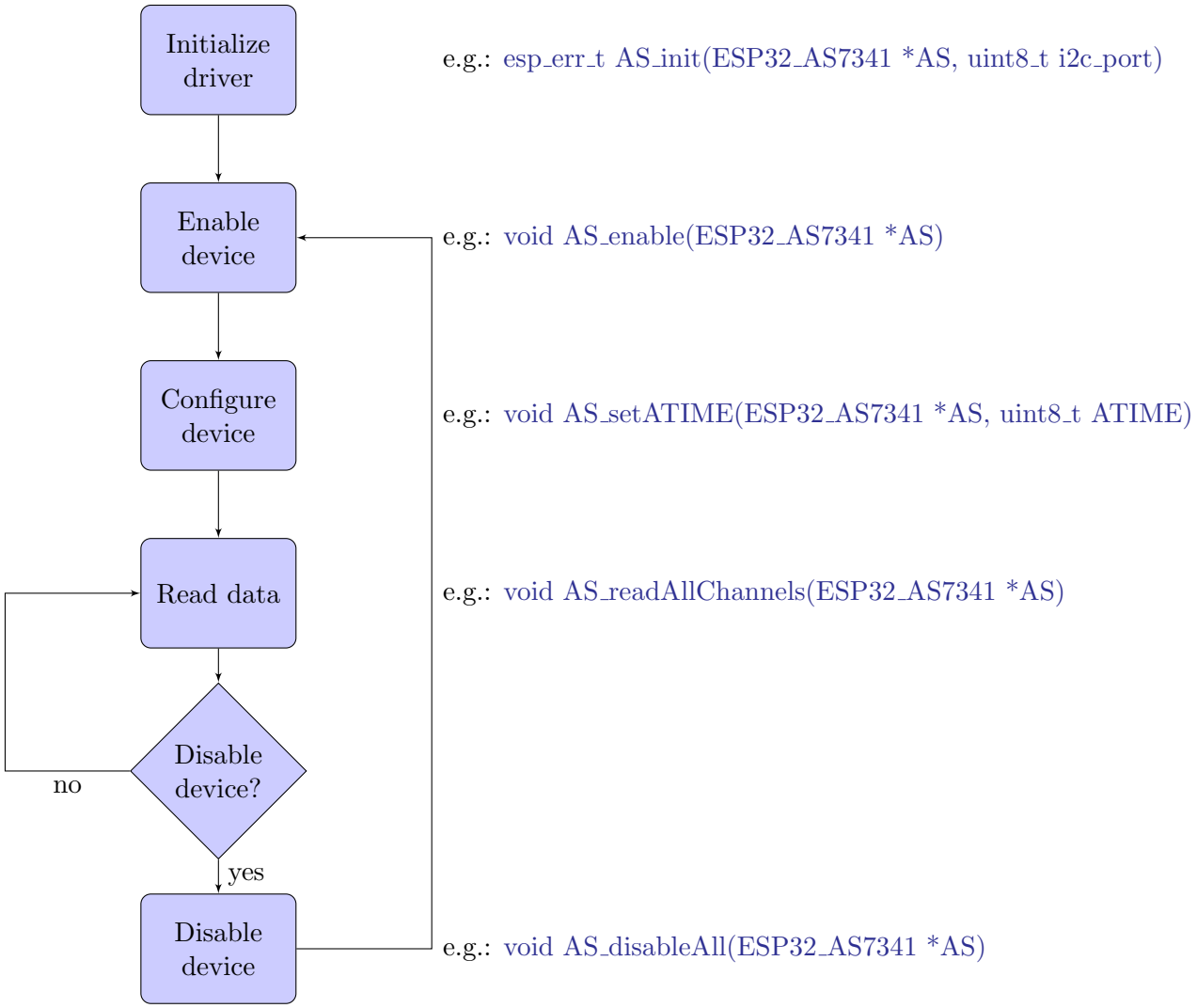
Every driver is written in a way that it **stores configurations in a structure**. When a HAL function is called, the first argument is always the pointer to the structure. An example of the structure could be the following:

Listing 4.1: AS7341 Driver Structure

```
typedef struct{
  uint8_t i2c_port;
  bool init;
  bool enabled;
  as7341_mode_t mode;
  as7341_gain_t gain;
  as7341_int_cycle_count_t APERS;
  uint8_t ATIME;
  uint16_t ASTEP;
  uint16_t HT;                  //Int high threshold
  uint16_t LT;                  //Int low threshold
  as7341_adc_channel_t TH_CH;   //Int threshold channel
  as7341_gpio_dir_t gpio_dir;
  bool gpio_inv;                //GPIO Inverted state
  bool out_gpio_value;
  bool reg_bank;
  uint16_t data[12];
} ESP32_AS7341;
```

HAL functions

All LAITS drivers HALs are designed to follow approximately the logic described in the next figure:



1. The initialize block consists in the **initialization of the default parameters** of the device structure and in the **acknowledgment of the device**: if the *device ID* is wrong the function will return an error, otherwise not.

2. Since most of the devices considered have the possibility to **enable and disable themselves to consume less power**, the first function that has to be called before interacting with device is the enable function. (In some cases the enable function is also called inside the "init" function).

3. Before reading data from the device, it **needs to be configured correctly**: for example we would set the integration time of the AS7341 or the maximum battery voltage of the MAX77960. To do this, multiple configuration functions are provided in order to set-up the correct register.

4. After the previous steps, the device is **ready to start measures**.

5. Finally, after the reading, the **device can be disabled to save energy**.

4.2.3 RTOS Tasks

LAITS avails itself of the flexibility given by the ESP multi-core *FreeRTOS* implementation to handle multiple tasks. FreeRTOS is an *MIT Licensed* ***Real Time Operating System*** **for embedded devices** supported by *Amazon AWS*.
FreeRTOS supports ***Pre-emptive scheduling*** as well as ***Co-operative scheduling***.

LAITS project defines a task source file for each peripheral, for the MQTT connection, for the microSD Card logging, for Wi-Fi connection and for OTA Updates.

Peripheral tasks

Every peripheral task file includes the respective peripheral driver and has at least two public functions: a ***start function*** and a ***task function***. Full device initialization (initialization, enable and configuration) is handled by the start function, called once in main task. Task function handles instead data reading, data logging on USB COM port as well as SD card and MQTT tasks notifying. **Every task function is looped infinitely**. In the main program a FreeRTOS task with a specific priority will be launched for every peripheral task function.

MQTT task

Message Queuing Telemetry Transport (MQTT) is a **lightweight**, **publish-subscribe**, application-layer network protocol. It is designed for connections that have devices with resource constraints or limited network bandwidth. It must run over a transport protocol that provides ordered, lossless and bi-directional connections, typically *TCP/IP*.

This protocol defines two types of network entities: **a message broker** and **a number of clients**. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device that implements the protocol and connects to an MQTT broker over a network.

Information in MQTT protocol is organized in a **hierarchy of topics**. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

The protocol provides **username and password** fields in the *CONNECT* message for authentication. The client has the option to send a username and password when it connects to a broker. However, MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. Nevertheless, this can be provided by using **Transport Layer Security (TLS)** to encrypt and protect the transferred information against interception, modification or forgery. For simplicity reasons, on the first releases of LAITS, TLS encryption is not implemented although it could be easily integrated with the existing code, making sure that rule N°8 is followed.

LAITS utilizes MQTT V3.1.1 protocol to **periodically send sensors data with a microsecond-precision NTP-synchronized timestamp** coming from the RTC tasks to the MQTT broker, than in turn sends the received data to LAITS backend (see Chapter 6).

The MQTT task file defines:

- A *MQTT start* function;
- A *MQTT Event Handler*;
- A *Publisher Task* for every sensor task.

The start function prepares, initiates and starts MQTT client on the node.

The event handler describes in which way the node must answer to each MQTT event.

MQTT publishers tasks take care of sending data to the specified broker whenever they receive an event by respective sensor task (i.e. when the sensor has complete the readings). Every message also contains the timestamp in which the data has been collected, in this manner, any possible **connection latency will not affect data reliability**. MQTT publisher functions are assigned to a task in the main program and they are looped infinitely.

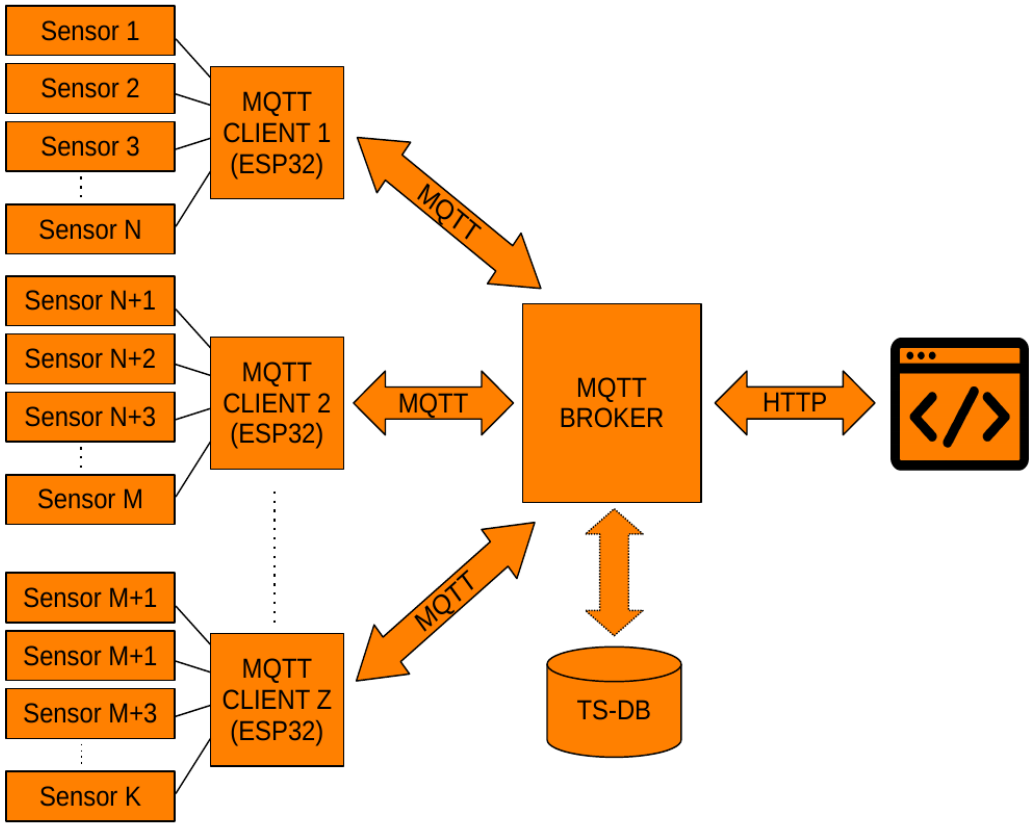


Figure 4.2: LAITS dataflow.

MicroSD task

As stated in Chapter 3.1.4, an SD card logging task is needed. This is easily handled with the aid of the ***FAT FS* library** provided in the ESP-IDF. Similarly to MQTT task file, SD task file contains:

- A *SD start* function;
- A *SD stop* function;
- A *main SD task* function;
- A *SD subtask* for each sensor task.

Since the SD card can be **removed or inserted at any time**, the filesystem must be respectively **unmounted** or **mounted**, respectively before and after these actions, in order not to corrupt data. This is done by the microSD *log enable button* provided in LAITS PCB: pressing one time the button will mount the filesystem (if the SD card is detected), pressing again will unmount it.

The SD main task is suspended until a specific interrupt wakes it. The interrupt is generated by the microSD log enable button. Whenever this task starts, it checks the current state of the SD card and toggles it calling stop or start functions, unmounting or mounting the FS. If the SD toggles to the mounted state, the *uSD Status LED* will turn on and all the **sensor substasks will be deployed**, otherwise the already existing **subtasks will be killed** and the LED will be turned off. The SD main task is assigned to a FreeRTOS task in the main function.

Every subtask logs **data together with its relative timestamp directly in *Line Protocol*** (see Chapter 6), in this way measures can be immediately loaded into the database without any processing needed.

Wi-Fi task

Wi-Fi task folder contains 3 source files:

- *wifi_task.c*
- *wifi_prov.h*
- *wifi_event_handler.h*

"*wifi_task.c*" is the main source file in which the **Wi-Fi initialization function** "*wifi_init_sta*" is defined. This function is called once in the fist lines of the main task to start Wi-Fi connection.

In order to comply with rules N°5,6,8, **Wi-Fi net credentials** such as SSID, and WPE, WPA, WPA2 or WPA3 password as well as **other secondary informations** such as MQTT broker IP address, port, MQTT username, MQTT password and so on are **provided through *Bluetooth LE*** via LAITS APP (see Chapter 5). Everytime the user wants to reset these fields, he/she can simply press the *Reset Provisioning button* and the node will appear as a Bluetooth device. The device will wait in provisioning mode until the provisioning has been completed, then it reboots in normal operation mode. **Data sent via Bluetooth are secured** using the security scheme: *Curve25519*-based key exchange, shared key derivation and *AES256-CTR* mode encryption of the data. Proof of Possession (PoP) string used to authorize the session and derive shared key. PoP string is simply hardcoded as a "*const char**" in first demo releases of LAITS firmware for simplicity reasons, but in next releases it will be moved into the ESP32 Non-Volatile Storage (NVS) **encrypted partition**. All these functionalities are implemented in the file "*wifi_prov.h*".

The file "*wifi_event_handler.h*" contains the event handler that calls the respective functions when specific Wi-Fi events are triggered.

OTA Updates task

An Over The Air (OTA) update is the **wireless delivery of new firmware**. OTA updates can be extremely useful for many reasons; two of them could be the fact that **multiples devices can be updated at the same time** and the fact that the device become capable of being **programmed without direct access** to it. ESP-IDF offers the possibility to update the firmware with a simple API call: "*esp_https_ota*". Using ESP-IDF HTTP libraries and a lightweight implementation of a JSON parser, *cJSON*, also integrated in Espressif SDK, it is possible to **check if a newer firmware version exists** in a specific URL. In that case the ESP will download the image and starts the upgrade.

To do this, a **custom partition table** must be loaded before flashing the firmware to the micro-controller. The partition table utilized for LAITS can be viewed in the attachment section A.4 LAITS-Firmware Partition Table.

4.2.4 Defines

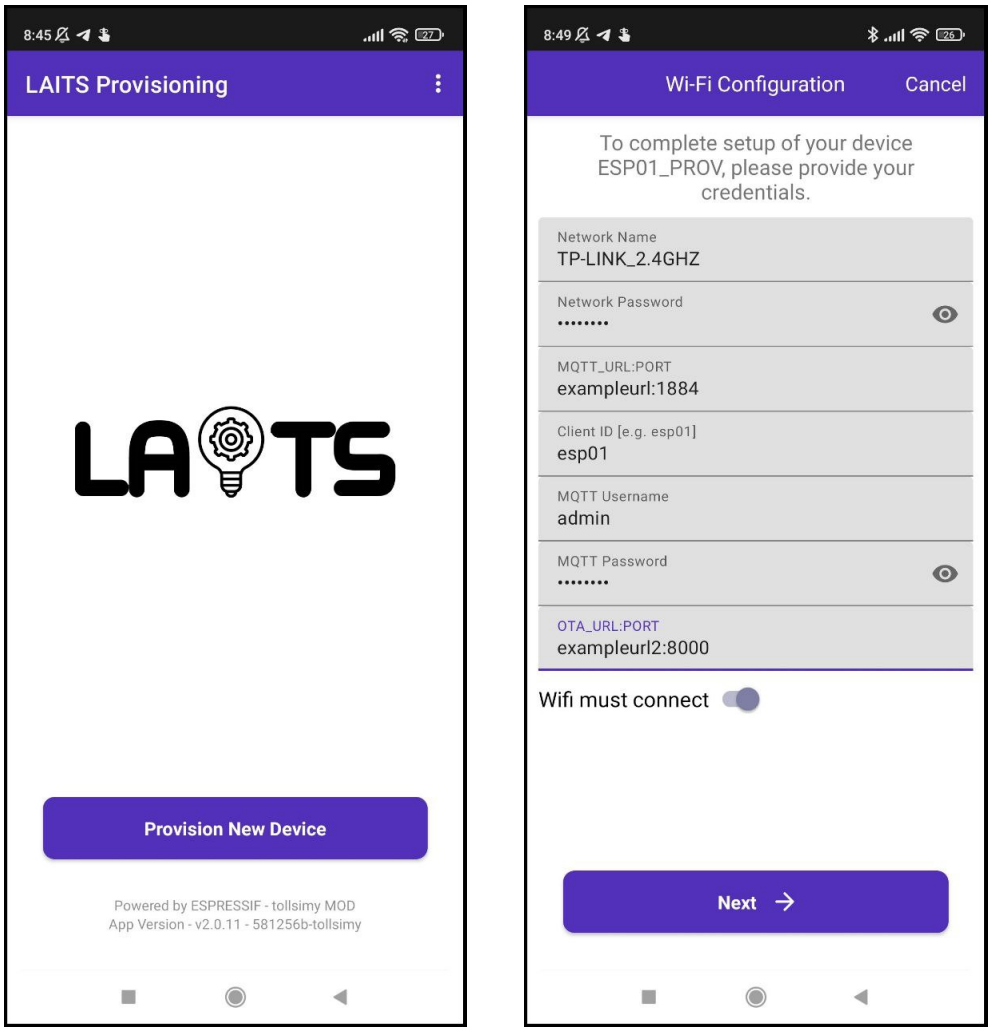
"*LAITS-defines.h*" header file contains the **definitions of all the pins of the board** as well as simple preprocessor directives to **enable or disable modules** based on the peripheral that are installed on the PCB. If, for example, the board has got the AMS7341 spectral sensor but it is not provided with TCS34725 nor BH1750 nor TSL2561, disabling their respective modules FreeRTOS tasks of those three devices will not be compiled.

4.2.5 Main

"*LAITS-main.c*" contains the function that is launched as the main FreeRTOS task and it handles the initialization of I2C buses, the definition of interrupts handlers, the verbosity level of the USB CDC log, the base configuration of the ESP GPIOs, peripherals initialization functions and the deployment of all the other tasks.

5 LAITS Provisioning APP

The *Android* application used for the provisioning is a custom version of the *ESP-IDF Provisioning App* provided by Espressif. LAITS-APP, in fact, implements a custom endpoint used to send custom data to the device to be provisioned such as already mentioned MQTT URL, MQTT port, MQTT username etc.



(a) (b)

Figure 5.1: Screenshots of LAITS Android APP.

Whenever the ESP32 boots-up in *Provisioning Mode* it starts to send Bluetooth Low Energy (BLE) **advertising** with so called Generic Access Profile (GAP) services **in broadcast mode**. In this way any listening device can see LAITS board as a BLE device. Once a device has established a connection with the ESP (normally called "*pairing*"), Generic Attribute Profile (GATT) services are used to **send data** (in this case the provisioning information) **only with the paired device**. Once the device is paired with another, GAP services and advertising stop, making the device not visible anymore. If there is the need to change for example the Wi-Fi network or the MQTT broker URL, LAITS can be booted again in provisioning mode pressing the *Reset Provisioning button*.

6 Server-Side

Messages received by the MQTT broker need to be stored and the user should be able to visualize them whenever he/she wants. To do so, a dedicated server must be used.
This chapter is divided in two sections: *Backend* and *Frontend*. Backend section explains all the tools that reside in the actual server implemented to store the received strings while frontend section talks about the tool adopted to visualize data and render graphical charts. In addition to that, backend section also contains the implementation of the OTA updates server.

6.1 Backend

6.1.1 MQTT Broker

As already mentioned, in order to receive messages from publishers and forward them to subscribers, MQTT needs a so called *broker*. The LAITS Project uses an open-source MQTT broker called *Eclipse Mosquitto*. Mosquitto can be easily installed in *Windows*, *Linux* and *Mac* environments. After specifying a configuration file, Mosquitto can be launched and it will be immediately active. A useful debug tool used to view MQTT messages, search topics, plot numeric topics and a lot more is *MQTT Explorer*, an open source "*swiss-army-knife*" for the mentioned protocol.

6.1.2 Time Series Database

Then, after the broker receives a message, the captured data need to be properly stored in a database. Since recorded data need to be precisely time-referenced, a Time Series Database (TSDB) is perfectly suited for the needs.

First tests were made using *Prometheus*, an open-source time series database. Prometheus functionalities were good but there was a conceptual problem: it is a **pull-type**, status-driven TSDB, meaning that the monitoring agent of the DB polls the targets periodically. For our purpose the optimal solution came after the switch to *InfluxDB*: an open-source **push-type**, event-driven TSDB. In this way, instead of polling for the data, metrics are pushed to the DB. To this extent, **push interval and in some way network latency are defined by the sender**.

6.1.3 MQTT Parser

In between the MQTT broker and the TSDB there is a custom parser. The role of this parser is **translating the received MQTT messages** into Line Protocol, the native syntax of InfluxDB metrics. The parser is written in *Python* and utilize *Eclipse Paho*, a python MQTT client implementation, to subscribe to all topics and the InfluxDB APIs to push data to the database. When a message is received, the parser converts its string into Line Protocol format and forwards the "new" message to the database.

6.1.4 OTA Updates Server

A simple demo implementation of the OTA Update server can be done using the python command: ***python3 -m http.server 8000*** to launch an HTTP server in the current directory using port 8000.

Again, for simplicity reasons, OTA updates are implemented using HTTP protocol although with simple steps they could be extended to HyperText Transfer Protocol Secure (HTTPS).

An important point need to be mentioned here: since all the above software are really lightweight, they can be be installed not only in a "real" server but also in a small *Raspberry PI*, making sure that the necessary equipment to be brought into the field is **light and compact** as required by rule

N°5 of LAITS concepts. Moreover, a **containerized *Docker* *Ubuntu*-based image** with all the requirements and relatives dependencies of the LAITS server is available.

6.2 Frontend

A frontend data visualization tool is imperative whenever the user wants to see previous recorded as well as real-time data. To simplify this process another open-source application is used: *Grafana*, an interactive web application that allows to build custom charts and graphs, simplifying data analytics. Grafana can be configured to retrieve data querying the existing InfluxDB database using its native query language: *Flux*.

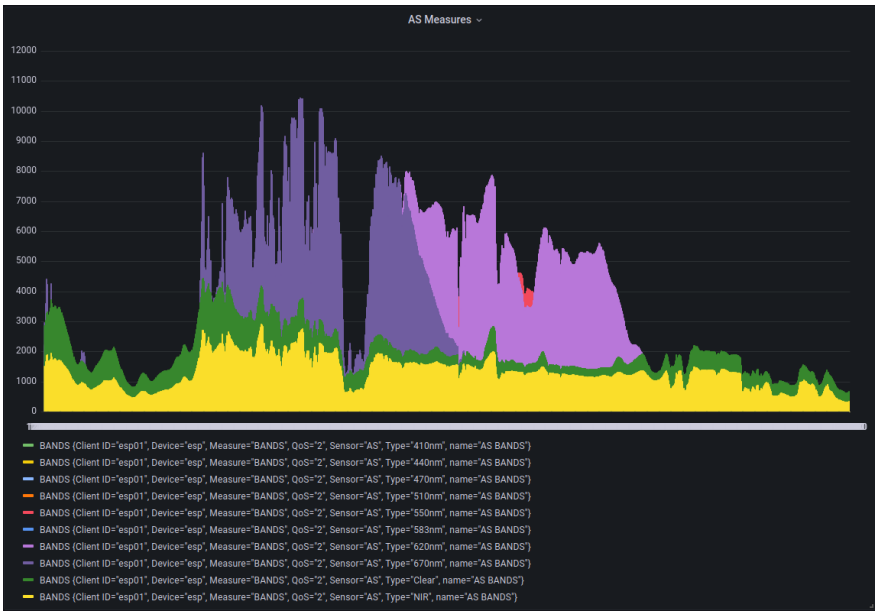


Figure 6.1: Example of a Grafana graph for LAITS project.

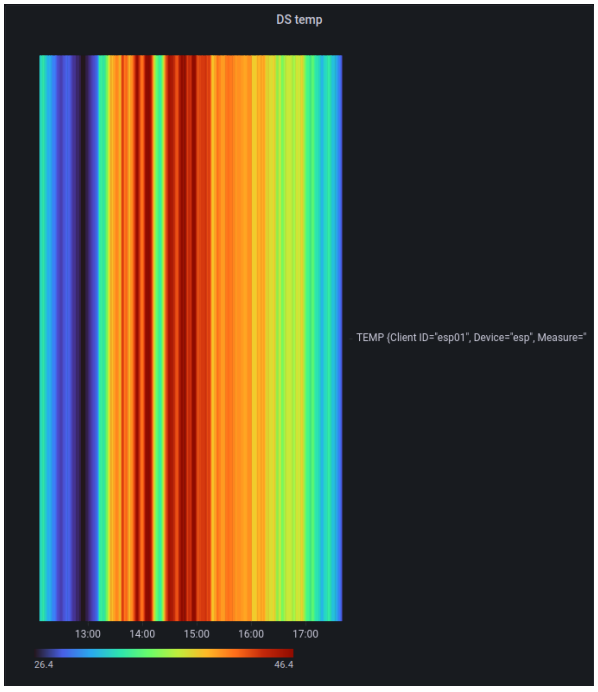


Figure 6.2: Another example of a Grafana graph for LAITS project.

7 Results

The aim of this eight chapter of the thesis is to discuss the results collected from all the aspects of the design.

In terms of hardware performances the board is really promising: 8 hours of data recording with Wi-Fi and MQTT enabled as well as SD logging, consumed less than 20% of the two Li-Ion 18650 batteries. If the board is put in deepsleep mode, current consumption drops below 250μA, guaranteeing weeks or maybe months of battery life.
Batteries recharge can be pretty fast if desired and it has reported no problem even with charge currents of more than 1A.

Despite some already fixed errors in first revisions of the PCB, the board is fully working and the strict layout rewards signals integrity: nor USB nor I2C buses shows any kind of noise or interference. Voltage levels of both switching and linear regulators are really stable and shows negligible noise.
Wi-Fi range with PCB antenna is very good even without high-power amplification.

After hours of testing the board have not shown overheating or any kind of problems (apart from small bugs related with the firmware).

The ease of use of the tool is clearly improved by the android provisioning APP, by the USB CDC debug messages, by fast bidirectional interaction with the user through LEDs and buttons and finally with the aid of the Docker container for the Server-Side software. The presence of OTA updates gives the possibility to maintain and improve the security of the device very easily.

For what concern data acquisition it is possible to state that the sampling rate, despite being tested up to an already overkill frequency of 1Hz, can be pushed far beyond the required values.

In Figure 8.1 it is possible to see the results of a 5 hours continuous demo recording:

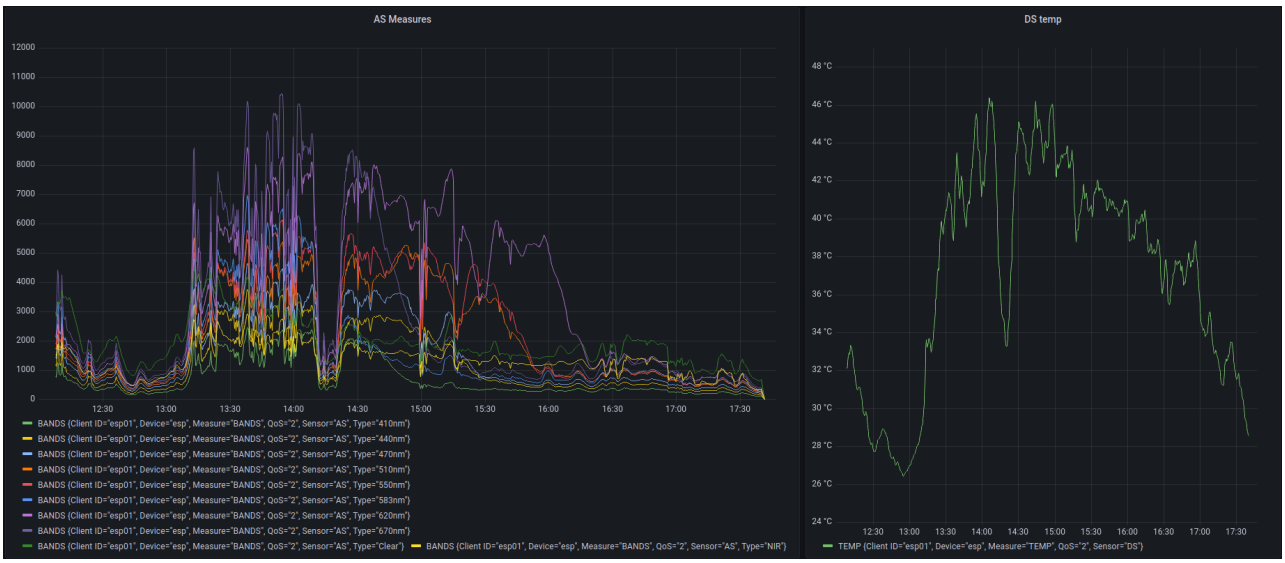


Figure 7.1: Results of data acquisition in a cloudy afternoon of September.

NOTE: temperature values are not reliable since the demo recording was made without the enclosure and so the DS18B20 was directly exposed to sunlight.
NOTE 2: none of the followings data is intended as absolute calibrated.

8 Conclusions and Future Work

The *Light Analysis Iot porTable Spectrometer* was developed to solve a challenging problem that prevents the extraction of radiometrically correct data from UAV-based spatially-scanned hyperspectral images: the **variation in time domain of the *Spectral Radiance Received by the interested surface* during the acquisition**.

Successful results has been achieved and the custom developed PCB, along with the open-source designed firmware and the server-side Docker image is able to cover a big piece of this challenge. In fact, all the data acquisition, storing and visualization stack, despite small bugs and not yet implemented features, is fully usable. Of course the provided sensors need absolute calibration to properly compare measurements, however, the analysis of the obtained results leads to the following conclusion: an affordable spectrometer is able to capture relevant data for our scope.
None of the established requirements has been overlooked and the solution is **up-to-date with IoT standards**, in fact, it provides the ability of running multiple nodes, it provides **security standards** such as authentication and data encryption, and lastly, secondary features give the user an extremely valuable handiness. Neither the cost has been left out, in fact the total price of this solution is not even comparable with one-tenth of the price of an hyperspectral camera, giving the possibility to integrate this tool with existing configurations **without affecting too much the total budget**.

Future research should certainly further test whether the tool can be adapted to **capture data in an optimal way even from the UAV** itself without suffering from tilt and movements on the unmanned aircraft. A possible parameter to work with for this purpose could be the Field Of View (FOV) of the sensor. If necessary, aircraft Inertial Measurement Unit (IMU) data could be used to retrieve a time-referenced 3D orientation of the sensor, furthermore, the expansion connector provided in the top of LAITS hardware is capable of connecting a dedicated I2C accelerometer, gyroscope or even a small IMU.

Future research should also further develop and confirm these initial findings by **exploiting the information that this instrumentation can supply** to actually compensate irradiance variation and especially moving cloud shadows in real hyperspectral images taken from Unmanned Aerial Vehicles.

Bibliography

[1] Nicola Acito and Marco Diani. Unsupervised atmospheric compensation of airborne hyperspectral images in the vnir spectral range. *IEEE Transactions on Geoscience and Remote Sensing*, 56(4): 2083–2106, 2018. doi: 10.1109/TGRS.2017.2774360.

[2] *11-Channel Spectral Sensor Frontend*. AMS, 2018. URL `https://www.mouser.com/catalog/specsheets/AMS_03152019_AS7341_DS000504_1-00.pdf`.

[3] David Antliff. Esp32-compatible example for maxim integrated ds18b20 programmable resolution 1-wire digital thermometer, 2017. URL `https://github.com/DavidAntliff/esp32-ds18b20-example`.

[4] ReSe Applications. Atcor-4 user guide. `https://www.rese-apps.com/pdf/atcor4_manual.pdf`, September 2021.

[5] Burdette Barker, Wayne Woldt, Brian Wardlow, Christopher Neale, Mitchell Maguire, Bryan Leavitt, and Derek Heeren. Calibration of a common shortwave multispectral camera system for quantitative agricultural applications. *Precision Agriculture*, 21, 08 2020. doi: 10.1007/s11119-019-09701-6.

[6] Ulrich Beisl. Reflectance calibration scheme for airborne frame camera images. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 39:1–5, 07 2012. doi: 10.5194/isprsarchives-XXXIX-B7-1-2012.

[7] Lorenzo Bruzzone. Corso di telerilevamento e radar. Telerilevamento e Radar UniTN course, 2021.

[8] Danang Surya Candra, Stuart Phinn, and Peter Scarth. Cloud and cloud shadow removal of landsat 8 images using multitemporal cloud removal method. In *2017 6th International Conference on Agro-Geoinformatics*, pages 1–5, 2017. doi: 10.1109/Agro-Geoinformatics.2017.8047007.

[9] Renaud Cerrato. Pcf2127 rtc driver, 2013. URL `https://github.com/torvalds/linux/blob/master/drivers/rtc/rtc-pcf2127.c`.

[10] Yong Chen, Wei He, Naoto Yokoya, and Ting-Zhu Huang. Total variation regularized low-rank sparsity decomposition for blind cloud and cloud shadow removal from multitemporal imagery. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 1970–1973, 2019. doi: 10.1109/IGARSS.2019.8900086.

[11] Docker. Source repo for docker's documentation, 2016. URL `https://github.com/docker/docker.github.io`.

[12] Eclipse. Eclipse mosquitto - an open source mqtt broker, 2014. URL `https://github.com/eclipse/mosquitto`.

[13] Eclipse. Eclipse paho python mqtt client library, 2014. URL `https://github.com/eclipse/paho.mqtt.python`.

[14] Espressif. Espressif iot development framework, 2016. URL `https://github.com/espressif/esp-idf`.

[15] Espressif. Android provisioning application for esp-idf unified provisioning, 2018. URL https://github.com/espressif/esp-idf-provisioning-android.

[16] T Fuyi, S. K Mohammed, K Abdullah, H. S Lim, and K.S. Ishola. A comparison of atmospheric correction techniques for environmental applications. In *2013 IEEE International Conference on Space Science and Communication (IconSpace)*, pages 233–237, 2013. doi: 10.1109/IconSpace.2013.6599471.

[17] Nathalie Gorretta, Xavier Hadoux, and Sylvain Jay. Multi-temporal hyperspectral data classification without explicit reflectance correction. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4228–4231, 2015. doi: 10.1109/IGARSS.2015.7326759.

[18] Grafana. Grafana: The open and composable observability and data visualization platform, 2014. URL https://github.com/grafana/grafana.

[19] Teemu Hakala, Lauri Markelin, Eija Honkavaara, Barry Scott, Theo Theocharous, Olli Nevalainen, Roope Näsi, Juha Suomalainen, Niko Viljanen, Claire Greenwell, and Nigel Fox. Direct reflectance measurements from drones: Sensor absolute radiometric calibration and system tests for forest reflectance characterization. *Sensors*, 18(5), 2018. ISSN 1424-8220. doi: 10.3390/s18051417. URL https://www.mdpi.com/1424-8220/18/5/1417.

[20] Henna-Reetta Hannula, Kirsikka Heinilä, Kristin Böttcher, Olli-Pekka Mattila, Miia Salminen, and Jouni Pulliainen. Laboratory, field, mast-borne and airborne spectral reflectance measurements of boreal landscape during spring. *Earth System Science Data Discussions*, pages 1–37, 08 2019. doi: 10.5194/essd-2019-88.

[21] Changmiao Hu, Lian-Zhi Huo, Zheng Zhang, and Ping Tang. Multi-temporal landsat data automatic cloud removal using poisson blending. *IEEE Access*, 8:46151–46161, 2020. doi: 10.1109/ACCESS.2020.2979291.

[22] Hyspex. Hyspex mjolnir v-1240 specs. https://www.hyspex.com/media/qdfpaitf/hyspex_mjolnir_v-1240.pdf, 2018.

[23] Nilton Imai, Antonio Tommaselli, Adilson Berveglieri, and Erika Moriya. Shadow detection in hyperspectral images acquired by uav. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13:371–377, 06 2019. doi: 10.5194/isprs-archives-XLII-2-W13-371-2019.

[24] Adafruit Industries. Driver for adafruit's tcs34725 rgb color sensor breakout, 2013. URL https://github.com/adafruit/Adafruit_TCS34725.

[25] Adafruit Industries. Adafruit as7341 arduino library, 2020. URL https://github.com/adafruit/Adafruit_AS7341.

[26] Influxdata. Influxdb: Scalable datastore for metrics, events, and real-time analytics, 2013. URL https://github.com/influxdata/influxdb.

[27] Saiqa Khan, Zainab Pirani, Taniya Fansupkar, and Umama Maghrabi. Shadow removal from digital images using multi-channel binarization and shadow matting. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 723–728, 2019. doi: 10.1109/I-SMAC47947.2019.9032447.

[28] KiCad. Source repo for kicad eda, 1992. URL https://gitlab.com/kicad/code/kicad.

[29] Xiao Liang, Xiang Ren, Zhengdong Zhang, and Yi Ma. Repairing sparse low-rank texture. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 482–495, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33715-4.

[30] J. Ian Lindsay. A library for the semtech sx150x family of i2c gpio expanders, 2020. URL https://github.com/jspark311/Arduino-SX150x.

[31] Xiaoxia Liu, Fengbao Yang, Hong Wei, and Min Gao. Shadow removal from uav images based on color and texture equalization compensation of local homogeneous regions. *Remote Sensing*, 14 (11), 2022. ISSN 2072-4292. doi: 10.3390/rs14112616. URL `https://www.mdpi.com/2072-4292/14/11/2616`.

[32] Yi Liu, José Bioucas-Dias, Jun Li, and Antonio Plaza. Hyperspectral cloud shadow removal based on linear unmixing. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 1000–1003, 2017. doi: 10.1109/IGARSS.2017.8127123.

[33] Steve Marschner. Radiometry notes. `https://www.cs.cornell.edu/courses/cs5625/2020sp/notes/radiometry-notes.pdf`, 2020.

[34] Ross McCluney. *Introduction to Radiometry and Photometry, Second Edition*. Artech House, 1994.

[35] Madhuri Nagare, Eiji Kaneko, Masato Toda, Hirofumi Aoki, and Masato Tsukada. Cloud shadow removal based on cloud transmittance estimation. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 4031–4034, 2018. doi: 10.1109/IGARSS.2018.8517580.

[36] Carl R. (Rod) Nave. Hyperphysics website. `http://hyperphysics.phy-astr.gsu.edu`, May 2017.

[37] Michael Kwok-Po Ng, Qiangqiang Yuan, Li Yan, and Jing Sun. An adaptive weighted tensor completion method for the recovery of remote sensing images with missing data. *IEEE Transactions on Geoscience and Remote Sensing*, 55(6):3367–3381, 2017. doi: 10.1109/TGRS.2017.2670021.

[38] R. Richter and A. Müller. De-shadowing of satellite/airborne imagery. *International Journal of Remote Sensing*, 26(15):3137–3148, 2005. doi: 10.1080/01431160500114664.

[39] Daniel Schläpfer and Rudolf Richter. Evaluation of brefcor brdf effects correction for hyspex, casi, and apex imaging spectroscopy data. In *2014 6th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, pages 1–4, 2014. doi: 10.1109/WHISPERS.2014.8077488.

[40] J.J. Simpson and J.R. Stitt. A procedure for the detection and removal of cloud shadow from avhrr data over land. *IEEE Transactions on Geoscience and Remote Sensing*, 36(3):880–897, 1998. doi: 10.1109/36.673680.

[41] Ruslan V. Uss. Component library for esp32, esp32-s2, esp32-c3 and esp8266 r, 2018. URL `https://github.com/UncleRus/esp-idf-lib`.

Acronyms

AFSI Attenuation Factor for the direct Solar Irradiance.
AIO All In One.
API Application Programming Interface.
BC Battery Charger.
BLE Bluetooth Low Energy.
CAD Computer-Aided Design.
CDC Communication Device Class.
CMOS Complementary Metal-Oxide Semiconductor.
DB Database.
DEM Digital Elevation Model.
DN Digital Number.
DRC Design Rule Checker.
DSM Digital Surface Model.
EDA Electronic Design Automation.
EM electromagnetic.
EMI Electromagnetic Interference.
ESD Electrostatic Discharge.
FDM Fused Deposition Modeling.
FOV Field Of View.
FS File System.
GAP Generic Access Profile.
GATT Generic Attribute Profile.
GND Ground.
GPIO General Purpose Input-Output.
GPS Global Positioning System.
GSI Ground Sample Interval.
HAL Hardware Abstraction Layer.
HTTP HyperText Transfer Protocol.
HTTPS HyperText Transfer Protocol Secure.
I2C Inter-Integrated Circuit.
IC Integrated Circuit.
IMU Inertial Measurement Unit.
IP Internet Protocol.
JSON JavaScript Object Notation.
LED Light Emitting Diode.
LiDAR Light Detection And Ranging.
LLVM Low Level Virtual Machine.
MCR Multi-temporal Cloud Removal.
MCU MicroController Unit.
ML Machine Learning.
MQTT Message Queuing Telemetry Transport.
NRND Non Recommended for New Designs.
NTP Network Time Protocol.
NVS Non-Volatile Storage.
OTA Over The Air.
PCB Printed Circuit Board.
PID Product ID.
pixel Picture Element.
PLA Polylactic Acid.
PoP Proof of Possession.
PSRAM Pseudo-Static Random-Access Memory.
RGB Red Green Blue.
RTC Real-Time Clock.
SD Secure Digital.
SDK Software Development Kit.
SLA Stereolithography.
SSID Service Set Identifier.
TCP Transmission Control Protocol.
TLS Transport Layer Security.
TSDB Time Series Database.
TX Receiver.
TX Transmitter.
UART Universal Asynchronous Receiver-Transmitter.
UAV Unmanned Aerial Vehicle.
URL Uniform Resource Locator.
USB Universal Serial Bus.
VID Vendor ID.
WPA Wi-Fi Protected Access.
WPE Wired Equivalent Privacy.

Glossary

achromatic optical diffuser Device which can diffuse light, i.e., essentially means strongly scrambling its wavefronts and reducing its spatial coherence. In other words, random or pseudo-random changes of optical phase for different parts of the special profile of incoming light is obtained. An achromatic diffuser maintains its characteristic across all the interested spectrum portion (usually referred as the visible one).

co-operative scheduling Type of scheduling mechanism in which every task **voluntarily** yield control (periodically, when idle or blocked).

datacube Synonym of n-D array.

FAT FS File system developed for personal computers in 1977. FAT32 is the 32bit cluster number variant.

irradiance Amount of light incoming to a certain point from possibly all directions.

keystone effect Optical aberration that strongly impacts the accuracy and the usability of pushbroom hyperspectral cameras: it can be seen as spatial misregistration of the spectrum.

n-D array n-Dimensional array.

pre-emptive scheduling Type of scheduling mechanism in which every task is periodically **interrupted** to switch to another one. Since each task has a priority assigned, whenever the switch occurs, other ready-to-run tasks will get the CPU time based on their priority.

relief effect Type of geometric distortion in which the image is further displaced as a result of relief displacement caused by the central projection of the aerial photograph.

smile effect Optical aberration that strongly impacts the accuracy and the usability of pushbroom hyperspectral cameras: it can be seen as a spectral shift of the sensor over its entire field of view (FOV).

solid angle The area of the segment of a unit sphere, centered at the apex, that the object covers.

stackup Arrangement of layers of copper and insulators that make up a PCB.

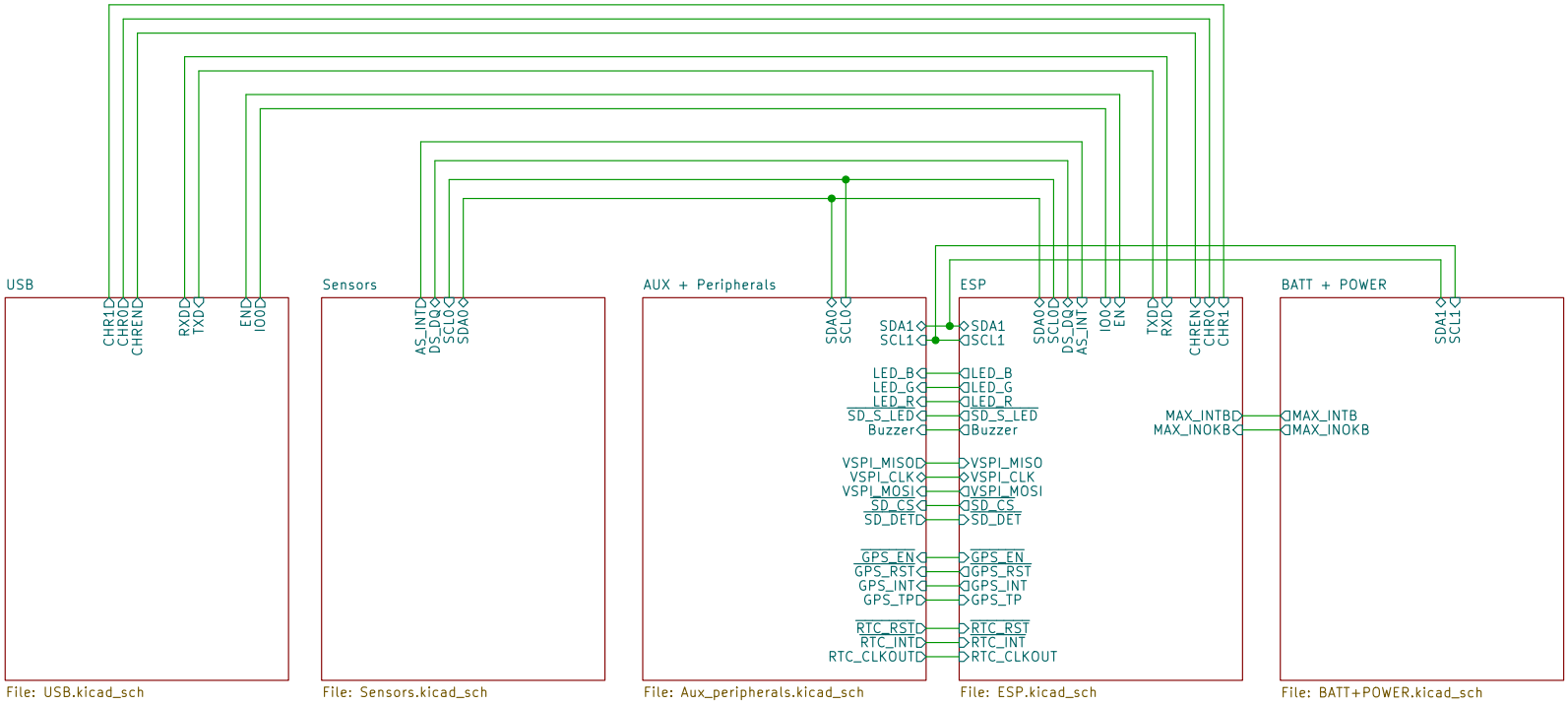
tilt effect Type of geometric distortion that causes image displacement radial from the isocenter on a flat plane.

VIA Two pads in corresponding positions on different copper layers of the board that are electrically connected by a hole through the board.

Appendix A

Attachments

A.1 LAITS-Hardware Schematics



Simone Tollardo

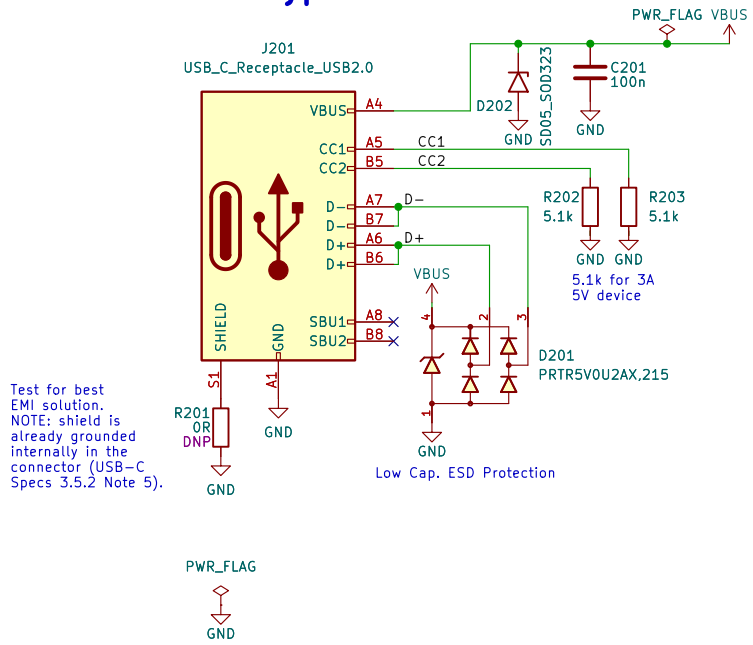
Sheet: /

File: LAITS-HW.kicad_sch

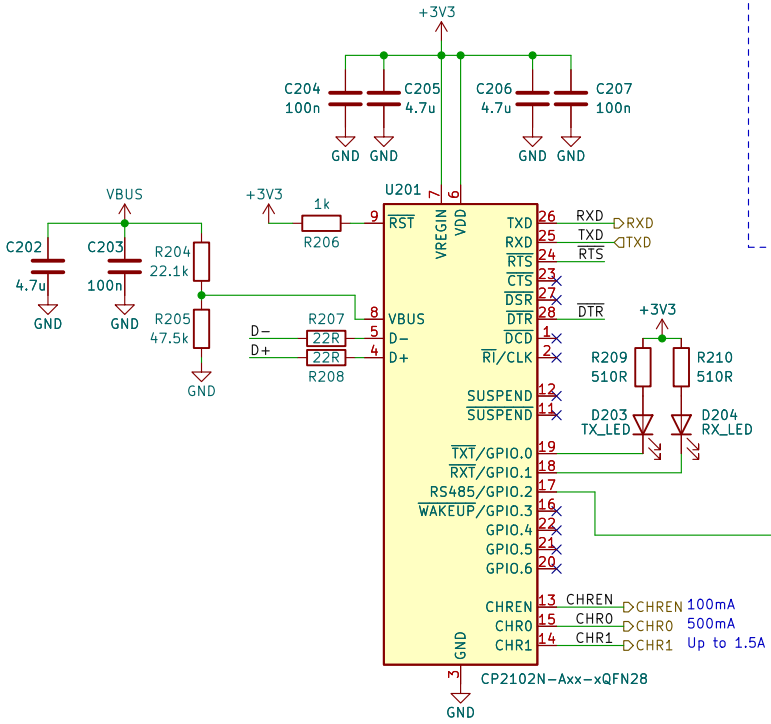
Title: LAITS (Light Analysis Iot porTable Spectrometer)

Size: A4	Date: 2022-08-12	**Rev: 1.0.3**
KiCad E.D.A. kicad 6.0.7-1.fc36		Id: 1/6

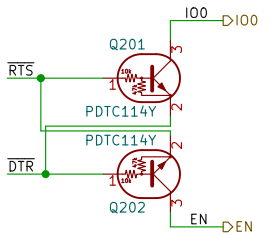
USB Type C

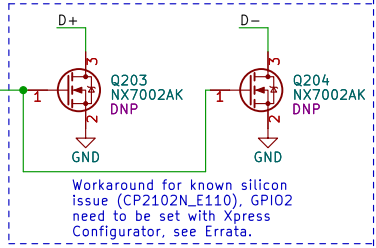


USB to UART



Bootloader Mode from USB





PWR_FLAG

GND

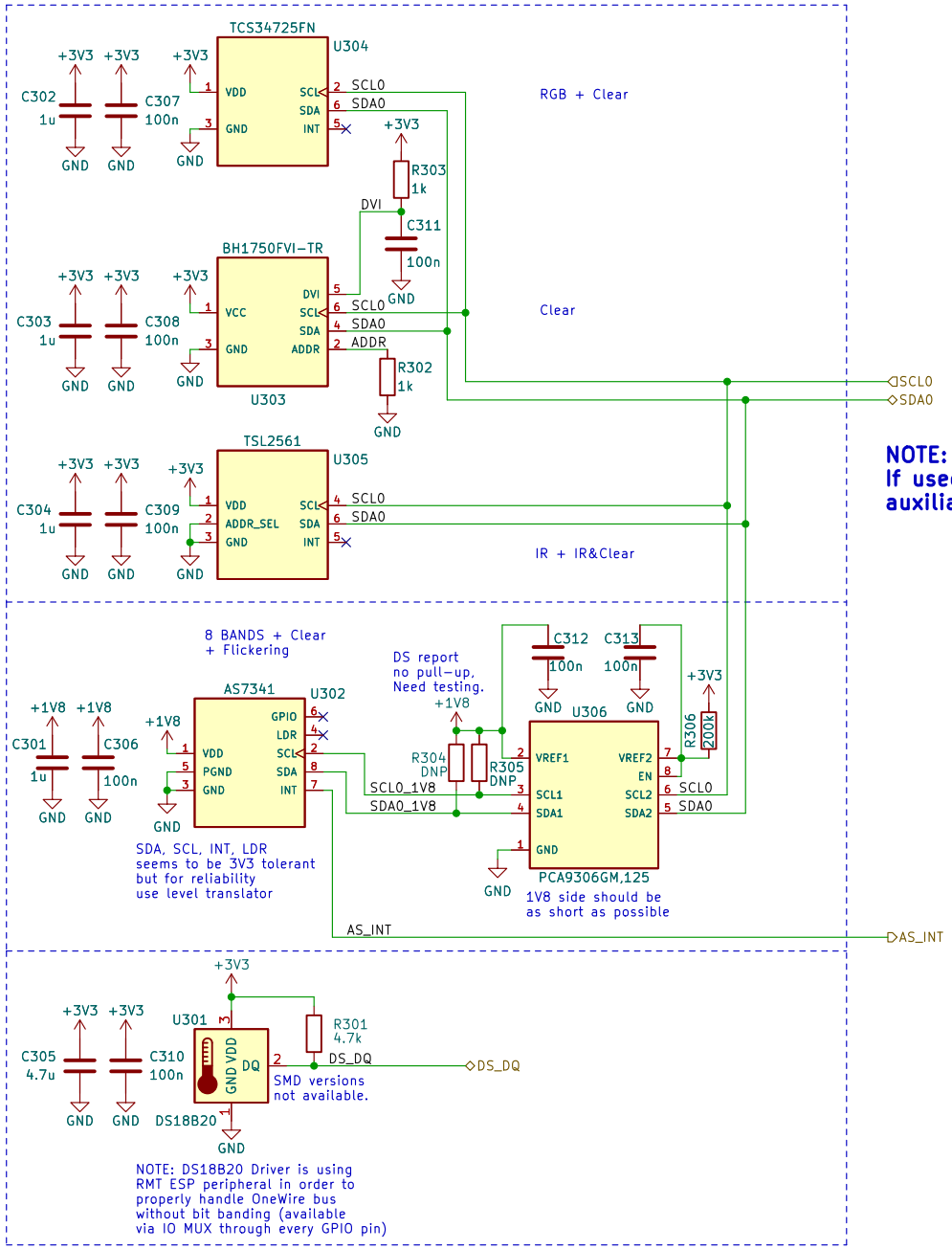
Simone Tollardo

Sheet: /USB/
File: USB.kicad_sch

Title: PS + USB

Size: A4	Date: 2022−08−12	**Rev: 1.0.3**
KiCad E.D.A. kicad 6.0.7−1.fc36		Id: 2/6

SENSORS



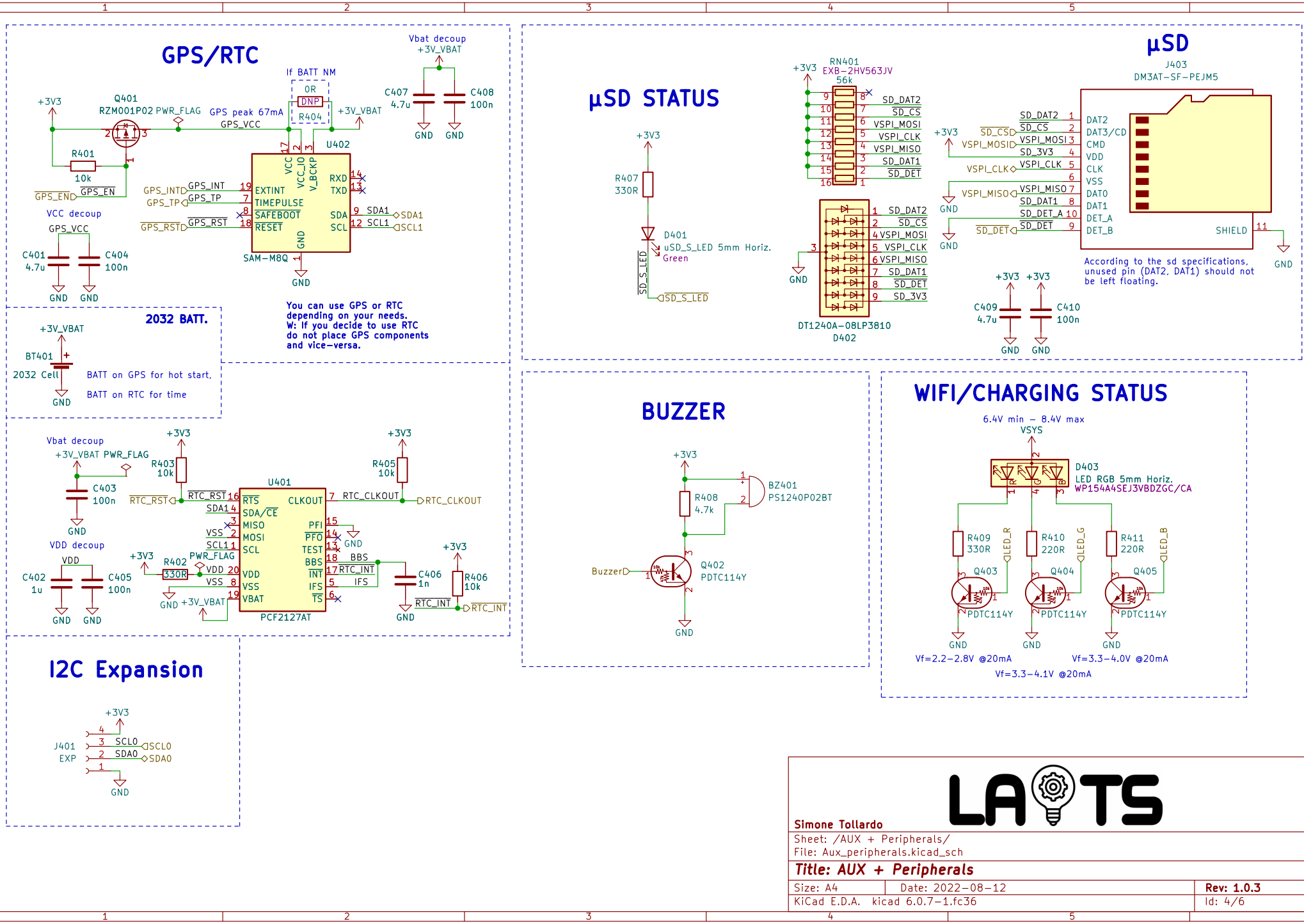
**NOTE: AS7341 can replace TCS, BH and TSL.
If used do not place TCS, BH, TSL and their
auxiliary components.**

Simone Tollardo

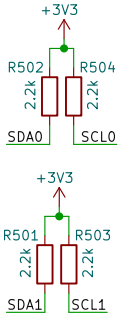
Sheet: /Sensors/
File: Sensors.kicad_sch

Title: Sensors

Size: A4	Date: 2022-08-12	**Rev: 1.0.3**
KiCad E.D.A. kicad 6.0.7-1.fc36		Id: 3/6

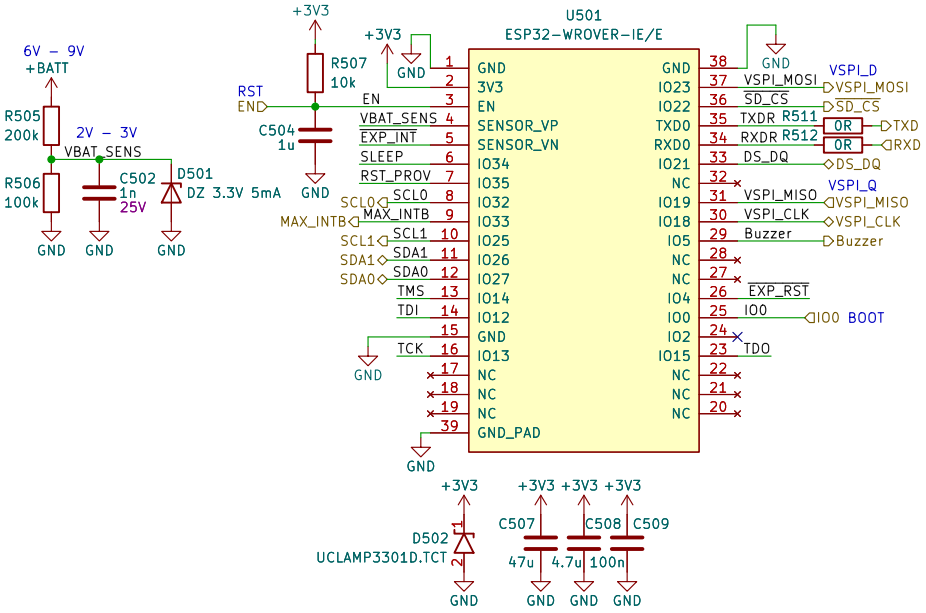

GPS/RTC
Vbat decoup
+3V_VBAT
If BATT NM
0R
DNP
R404
Q401
RZM001P02
PWR_FLAG
GPS peak 67mA
GPS_VCC
+3V_VBAT
C407
4.7u
C408
100n
GND
+3V3
R401
10k
GPS_EN
U402
VCC
VCC_IO
V_BCKP
GND
EXTINT
TIMEPULSE
SAFEBOOT
RESET
RXD
TXD
SDA
SCL
SDA1
SCL1
GPS_INT
GPS_TP
GPS_RST
SAM-M8Q
VCC decoup
C401
4.7u
C404
100n
You can use GPS or RTC depending on your needs.
W: If you decide to use RTC do not place GPS components and vice-versa.
2032 BATT.
BT401
2032 Cell
BATT on GPS for hot start,
BATT on RTC for time
Vbat decoup
PWR_FLAG
R403
10k
R405
10k
C403
100n
U401
RTC_RST
SDA1
VSS
SCL1
RTS
SDA/CE
MISO
MOSI
SCL
CLKOUT
PFI
PFO
TEST
BBS
INT
IFS
TS
VDD
VSS
VBAT
RTC_CLKOUT
RTC_INT
PCF2127AT
VDD decoup
C402
1u
C405
100n
R402
330R
C406
1n
R406
10k
I2C Expansion
J401
EXP
SCL0
SDA0
µSD STATUS
R407
330R
D401
uSD_S_LED 5mm Horiz.
Green
SD_S_LED
RN401
EXB-2HV563JV
56k
SD_DAT2
SD_CS
VSPI_MOSI
VSPI_CLK
VSPI_MISO
SD_DAT1
SD_DET
SD_3V3
DT1240A-08LP3810
D402
µSD
J403
DM3AT-SF-PEJM5
DAT2
DAT3/CD
CMD
VDD
CLK
VSS
DAT0
DAT1
DET_A
DET_B
SHIELD
According to the sd specifications, unused pin (DAT2, DAT1) should not be left floating.
C409
4.7u
C410
100n
BUZZER
R408
4.7k
BZ401
PS1240P02BT
Buzzer
Q402
PDTC114Y
WIFI/CHARGING STATUS
6.4V min - 8.4V max
VSYS
D403
LED RGB 5mm Horiz.
WP154A4SEJ3VBDZGC/CA
R409
330R
R410
220R
R411
220R
LED_R
LED_G
LED_B
Q403
Q404
Q405
PDTC114Y
Vf=2.2-2.8V @20mA
Vf=3.3-4.0V @20mA
Vf=3.3-4.1V @20mA
Simone Tollardo
Sheet: /AUX + Peripherals/
File: Aux_peripherals.kicad_sch
Title: AUX + Peripherals
Size: A4
Date: 2022-08-12
Rev: 1.0.3
KiCad E.D.A. kicad 6.0.7-1.fc36
Id: 4/6

I2C Pull-up



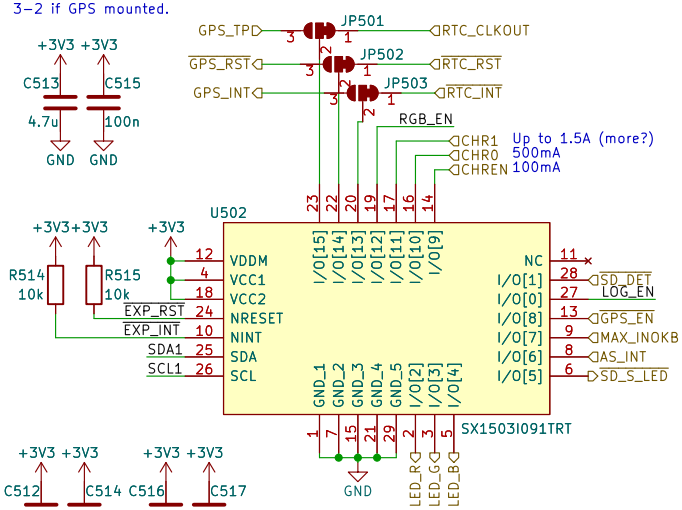
Depending on the I2C bus speed, these values may be adjusted.
100KHz=4,7K
400KHz=2,2K
In every case this need some testing.

ESP



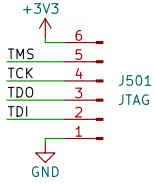
GPIO Expander

Solder Jumpers Pos.
1-2 if RTC mounted.
3-2 if GPS mounted.



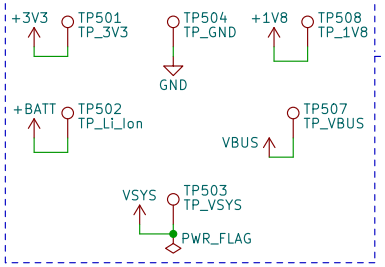
I/O 6,7,14,15 High Sink current 24mA.
Other pins 12mA sink current.
All pins 8mA source current.

JTAG



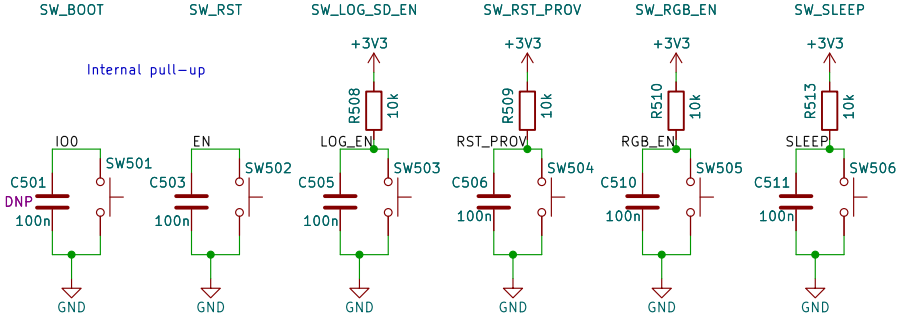
Since MTDI and MTDO are bootstrapping pins, wait for reset before using JTAG interface.

Testpoints



Do not place C501 to ensure proper startup sequence.

BUTTONS



ESP Bootstrapping pins:
MTDI (GPIO12) -> need to be kept low at startup: 0=LDO 3V3, 1=LDO 1V8

MTDO (GPIO15) -> 1=U0TXD Active
0=U0TXD Silent

BOOT (GPIO0) -> 1=SPI, 0=Bootloader

GPIO2 -> 0 for Bootloader mode

GPIO5 (Pull-up at startup) SDIO timing
Since we are not using SDIO interface we can freely use GPIO5.

GPIO34-35-36-37-38-39 Input only

RTC PINS can receive interrupt and wake up ESP from deep-sleep.

NINT, RGB, CHRG, RST PROV, MAX_INTB.

SPI and UART direct I/O via I/O MUX while I2C and other low speed stuff can be mapped anywhere through GPIO Matrix.

SPI, GPIO16, GPIO17 used by internal FLASH and PSRAM HSPI mapped on same pins of JTAG VSPI is free.

SPID=MOSI
SPIQ=MISO
SPICLK=SPICLK
SPIHD=Hold
SPIWP=Write Protect

WP & HD pin not used in SPI 1 bit mode

Simone Tollardo

Sheet: /ESP/
File: ESP.kicad_sch

Title: ESP

Size: A4	Date: 2022-08-12	**Rev: 1.0.3**
KiCad E.D.A. kicad 6.0.7-1.fc36		Id: 5/6

Battery Management

Energy Distribution Priority

- With a valid external power source at CHGIN:
  - The external power source is the primary source of energy.
  - The battery is the secondary source of energy.
  - Energy delivery to SYS has the highest priority.
  - Any remaining energy from the power source that is not required by the system is available to the battery charger.
- With no valid external power source at CHGIN:
  - The battery is the primary source of energy.
  - When OTG mode is enabled, energy delivery to SYS has the highest priority.
  - Any remaining energy from the battery that is not required by the system is available to power the CHGIN.

WARNING, NO REVERSE POLARITY PROTECTION!

Battery

+BATT, PWR_FLAG, +BATT
C2+, C2, C2-, B1-B2, BT601 2S_Battery 18650, C1+, C1, C1-
GND
2S1P 18650 Li-Ion.

Battery Charge LED

Led off charge off,
Led blinking charge on,
Led on when top-off treshold reached.

+3V3, R602 330R, D601 BATT_S_LED 5mm Horiz. Red, STAT

VBUS, R608 1206 10m, R605 10R, R609 10R, CSINP, C605 100n 10V, CSINN, C604 35V 15n, C606 35V 15n, GND, PGND
C607 10u 35v, C609 10u 35v, GND, PGND, GND
Ceramic X5R/X7R 1210

SCL1, SDA1, MAX_INTB, MAX_INOKB, PVL, R604 10k
R611 200k, R613 200k
4.7u C611 10V, 4.7R R616, SGND, GND
C613 4.7u 10V, SGND, GND
R610 10k, R612 10k, R614 10k, SGND, GND

U603
MAX77960BEFV12+

CHGIN	2	CHGIN	
CSINN	30	CSINN	
CSINP	29	CSINP	
PVL	27	PVL	
AVL	26	AVL	
SCL	14	SCL	
SDA	15	SDA	
INTB	18	INTB	
INOKB	17	INOKB	
STAT	16	STAT	
OTGEN	7	OTGEN	
DISQBAT	8	DISQBAT	
STBY	28	STBY	
GND	20	GND	
PGND	4	PGND	
CNFG	22	CNFG	
		BST1	1 BST1
		LX1	3 LX1
		LX2	5 LX2
		BST2	9 BST2
		SYS	6 VSYS
		SYSA	19
		BATT_1	12 +BATT
		THM	13 THM
		BATSP	11
		BATSN	10
		ISET	23 ISET
		ITO	25 ITO
		INLIM	21 INLIM
		VSET	24 VSET

0.22u 6.3V C614
L602 1.5u 5A Isat
C615 6.3V 0.22u

According to the DS, VSYS swings from ABOUT VBATT to VBATTREG= termination voltage=8.4V (Set via I2C or VSET resistor), so from about 6.4 to 8.4

VSYS, Ceramic X5R 1210
C618 47u 25V (16V), C619 47u 25V (16V)
GND PGND GND
C616 10u 25V Ceramic, C617 1u 25V
GND PGND, GND SGND

+BATT, As close as possible to battery terminals., GND PGND

Batt. Insertion Protection

D605 VSSAF3L45HM3_A/I
+BATT, VSYS, 3A

Temp Protection

AVL, R622 100k, THM, TH601 NTC 100k 4250, SGND, GND
Tcold = 4.1 °C
Thot = 53.2 °C

If at least one has no valid resistor value (ISET) charging is disabled by default and need to be set by I2C (choosen for safety reasons). ==>

PVL
JP601 INLIM, JP602 VSET, JP603 ISET, JP604 ITO
ATT!
INLIM_R, VSET_R, ISET_R, ITO_R

ITO_R	ISET_R	VSET_R	INLIM_R
R618 178k	R619 DNP	R620 86.6k	R621 54.9k
GND	GND	GND	GND SGND
200mA top-off		8.4V	1.5A Input max

USB BC 1.2 Imax = 1.5A.

According to the DS of CP2102N, implementing USB BC 1.2, 1.5A is the TOTAL MAX CURRENT drawn from USB 5V rail.

There are also some proprietary protocols (e.g. Apple, Samsung and Blackberry chargers) that allows to draw more than 1.5A without USB C PD, but they are not implemented on CP2102N.

3V3 Switching

VSYS, F601 MF-LSMF075X, FUSE_OUT, 0.75A Hold 1.5A Trip
D603 VSSAF3L45HM3_A/I, 3A
D602 SMAJ15A, GND
TVS VBR 17V IBR=1mA
Cer. low ESR C602 10u 25V, GND
PWR_FLAG, SW_VIN, R623 100k, EN
AP62301Z6-7: VIN, SW, BST, FB, EN, GND, U602
SW_NODE, L601 3.3u, SW_BST, C608 100n 6V3
Max 1.5A, SW_NODE_OUT, C610 22u 6V3, C612 22u 6V3, GND, Low ESR
R617 OR BUCK TEST CRCW12060000Z0EAHP, +3V3
SW601 SW_SPDT, GND
On/Off Switch.
OBV battery rail remains active.

Even if Batt. is not present, VSYS is always regulated to VBATTREG (see datasheet).

1V8 Linear

Needed only if AS7341 used.

+3V3, C601 1u, R601 10k, GND
U601: VIN, VOUT, EN, GND, PG
1V8_EN, GND NCV8164ASN180T1G
300mA max, +1V8, C603 10u, GND
R603 100k
SW_NODE_OUT, R606 31.6k 1%, SW_FB, R607 10k 1%, GND

PDmax=(Tjmax-Ta)/R0JA)=
(125-20)/158=0.66 W
PD=Vin(Ignd(Iout))+Iout(Vin-Vout)=
3.3(0)+300m(3.3-1.8)=0.45 W

+3V3, D604 3V3_LED Red, R615 510R, GND

3V3 Peripherals MAX absolute ratings:
- ESP32 = 500mA (min required by DS)
- GPIO Expander = 300mA (with all pin max output)
- CP2102N = 50mA
- 4 Sensors = 10mA
- GPS/RTC = 70mA/50mA
- Buzzer = 35mA

TOT < 1.5A -> Real probably < 0.7A

The regulator always priorities the system, the remaining power will be delivered to battery if charging. So, if charging at 1.5A and system is using 1A, battery charger will get only 0.5A.
If Input power is not enough to power the system, battery will output current to system. So, if available input power is 0.5A and system is using 1A, if battery are charged they will deliver the remaining 0.5A, otherwise the DCDC will detect a failure.
For this reason the ESP32 has as input CHREN, CHR0 and CHR1 and it should go to power save mode (disable wifi and other peripherals) if power is not enough.

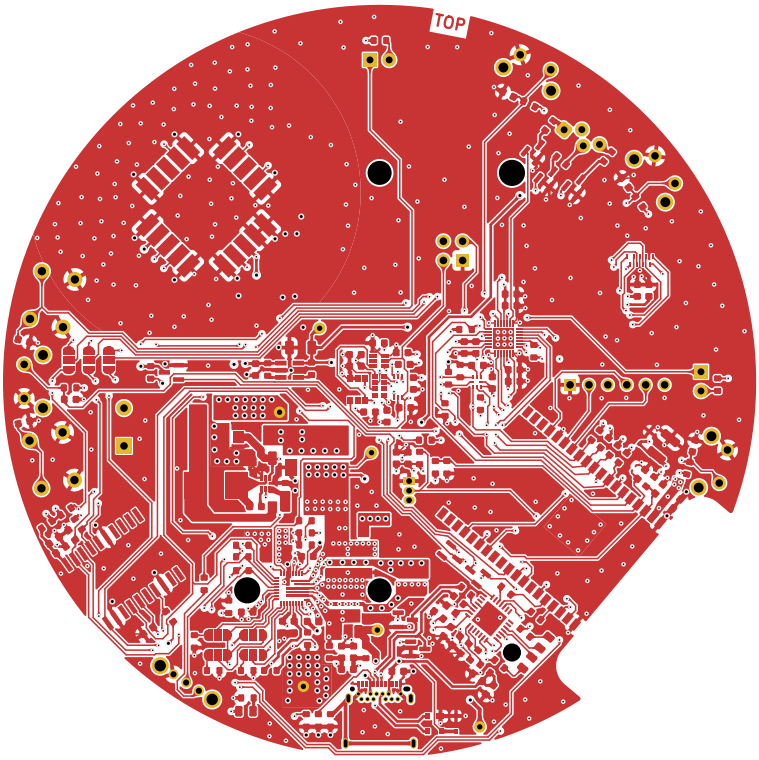
Simone Tollardo

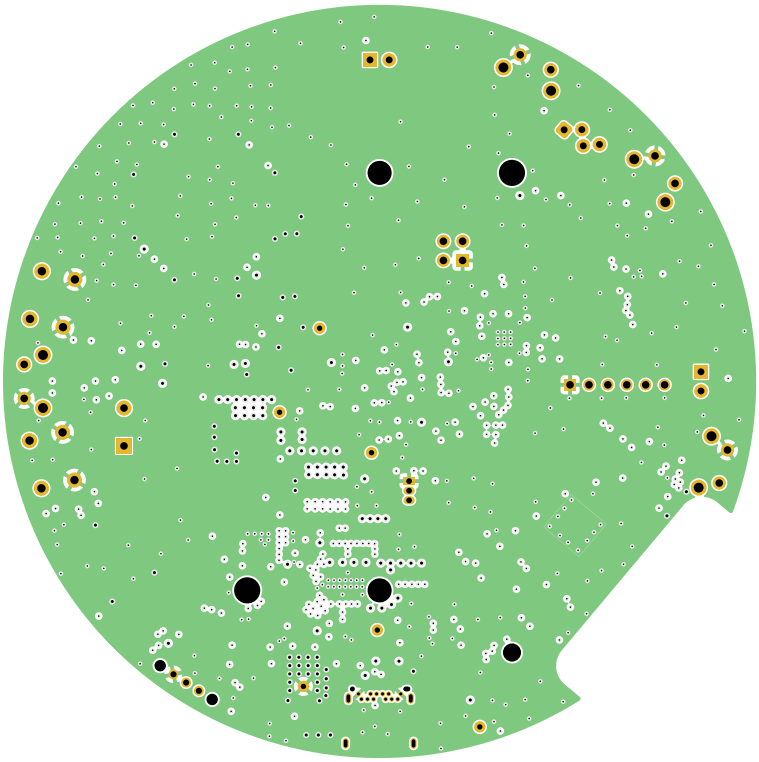
Sheet: /BATT + POWER/
File: BATT+POWER.kicad_sch

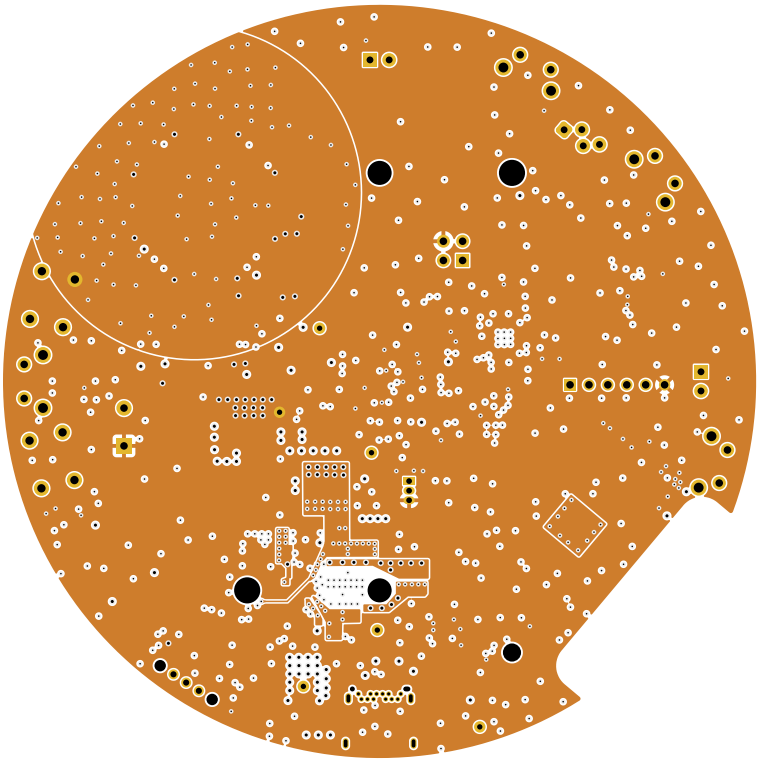
Title: BATT

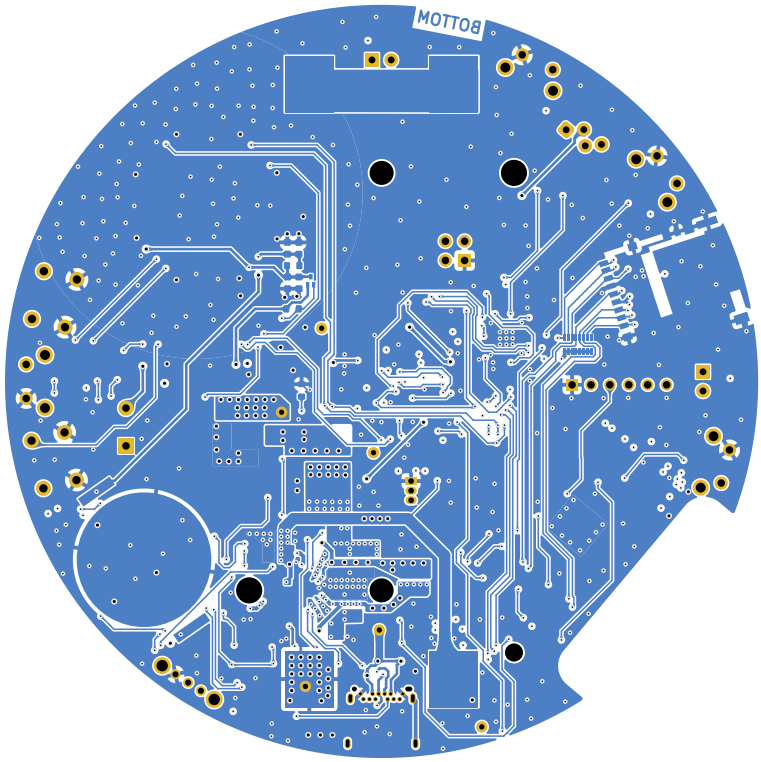
Size: A4	Date: 2022-08-12	**Rev: 1.0.3**
KiCad E.D.A. kicad 6.0.7-1.fc36		Id: 6/6

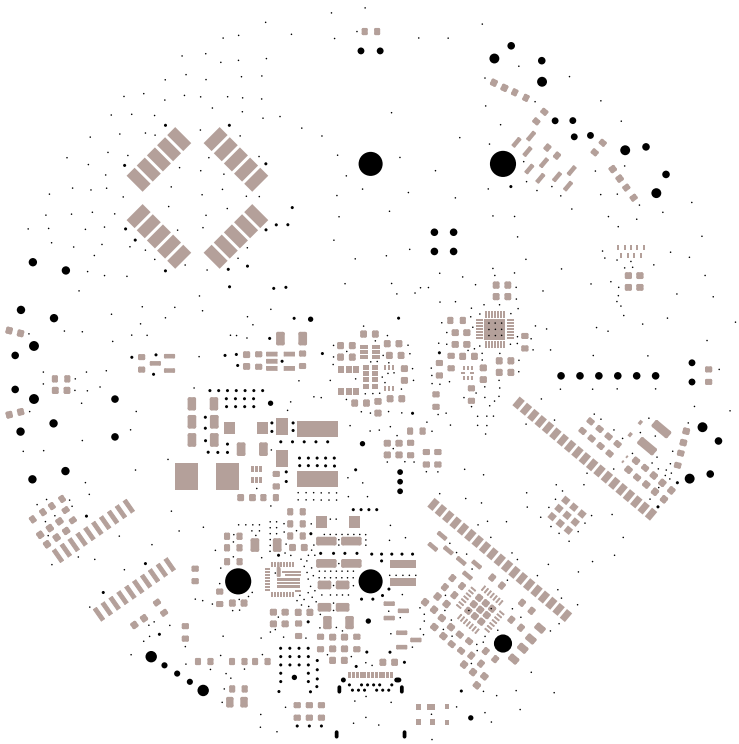
A.2 LAITS-Hardware Gerber layers

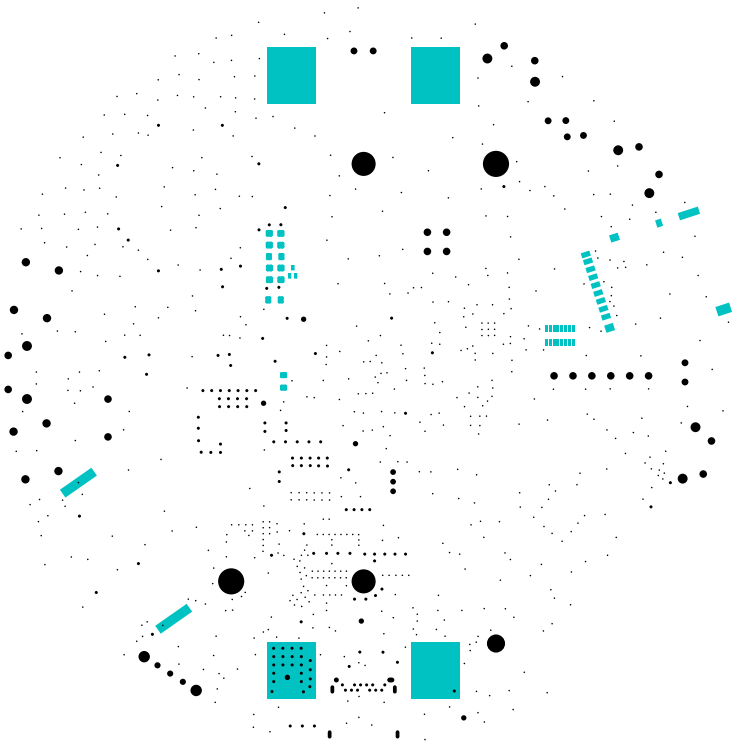

TOP

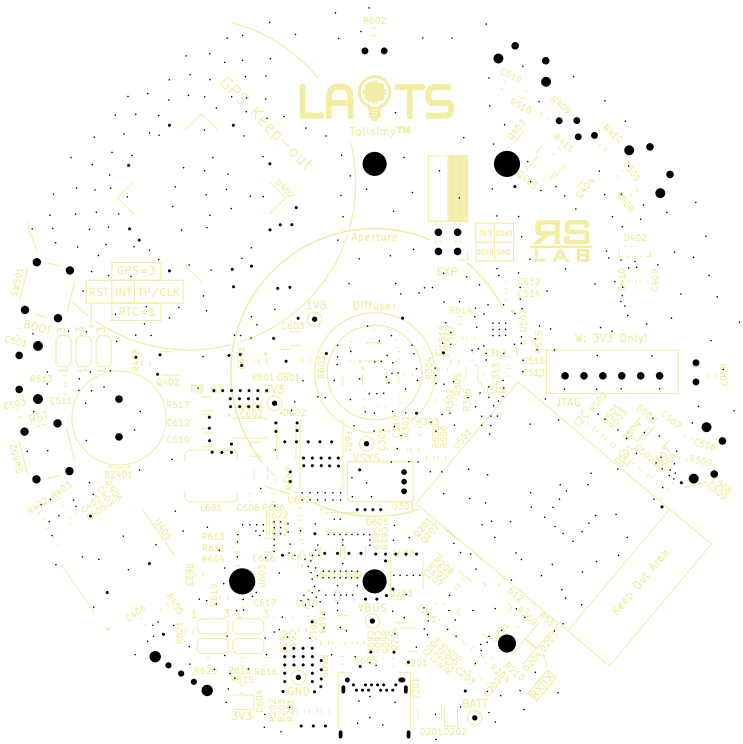


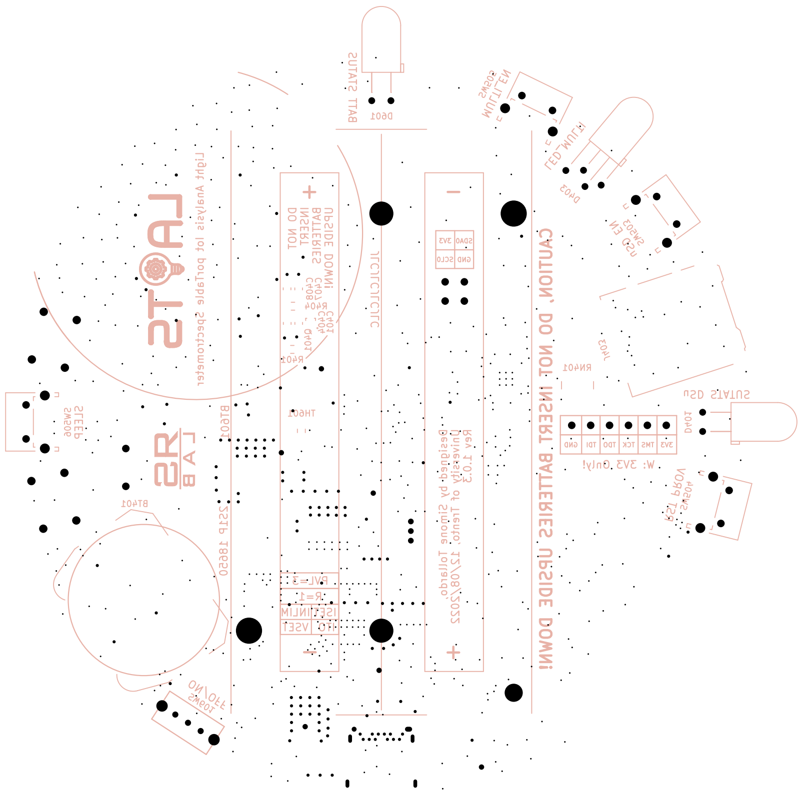


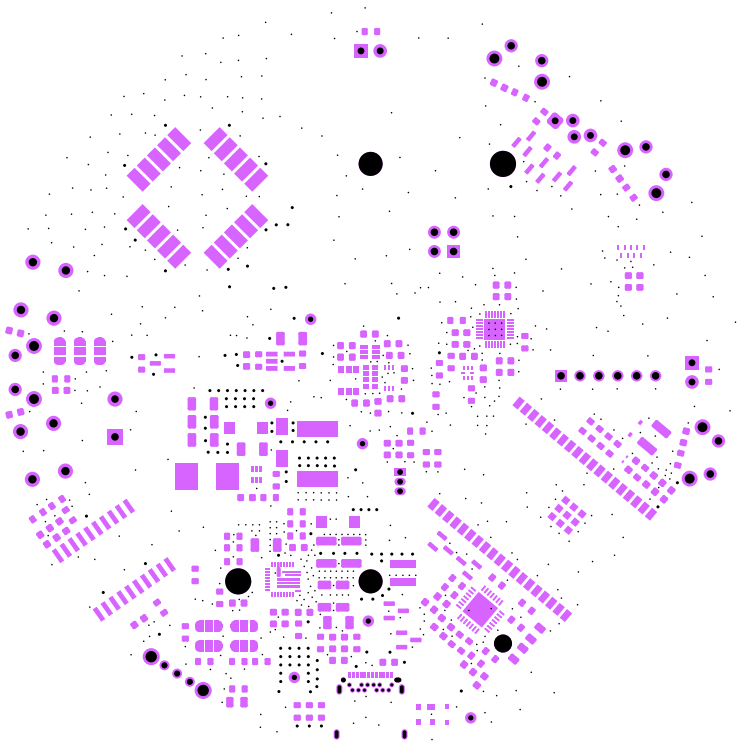

BOTTOM

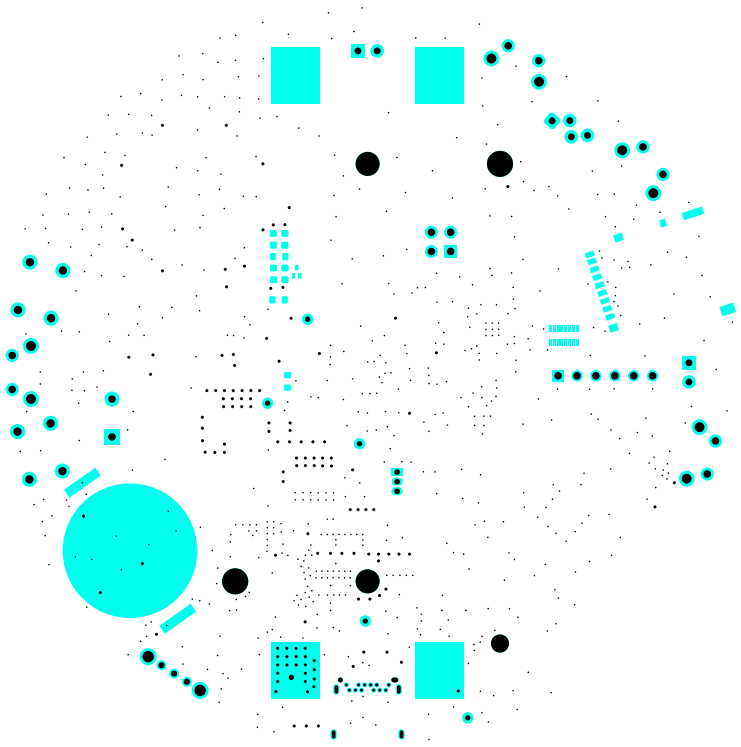





LAOTS
Talisimy™
GPS Keep-out
Aperture
Diffuser
EXP
3V3 SDA0 SCL0 GND
RS LAB
GPS=3
RST INT TP/CLK
RTC=1
1V8
3V3
W: 3V3 Only!
JTAG
Keep Out Area
BATT
GND


CAUTION, DO NOT INSERT BATTERIES UPSIDE DOWN!
Light Analysis for portable Spectrometer
Rev 1.0.3
BAT STATUS
LED WHITE
USB STATUS
ON/OFF
W: 3V3 Only!





A.3 LAITS-Hardware BOM

PCB Version: v1.0.3			
Date: 01/09/2022			
Component Count: 189			
Ref	Qnty	Value	Cmp name
"BT401, "	1	2032 Cell	Battery_Cell
"BT601, "	1	2S_Battery 18650	2S_Battery
"BZ401, "	1	PS1240P02BT	Buzzer
"C201, C203, C204, C207, C306, ..."	28	100n	C
"C202, C205, C206, C305, C401, ..."	13	4.7u	C
"C301, C302, C303, C304, C402, ... "	8	1u	C
"C406, C502, "	2	1n	C
"C507, C618, C619, "	3	47u	C
"C602, C603, C607, C609, C616, "	5	10u	C
"C604, C606, "	2	15n	C
"C610, C612, "	2	22u	C
"C614, C615, "	2	0.22u	C
"D201, "	1	PRTR5V0U2AX,215	PRTR5V0U2AX,215
"D202, "	1	SD05_SOD323	SM6T100A
"D203, "	1	TX_LED	LED
"D204, "	1	RX_LED	LED
"D401, "	1	uSD_S_LED 5mm	LED
"D402, "	1	DT1240A-08LP3810	DT1240A-08LP3810
"D403, "	1	LED RGB 5mm	LED_RGB
"D501, "	1	DZ 3.3V 5mA	D_Zener
"D502, "	1	UCLAMP3301D.TCT	TVS-Unidirectional
"D601, "	1	BATT_S_LED 5mm	LED
"D602, "	1	SMAJ15A	1.5KExxA
"D603, D605, "	2	VSSAF3L45HM3_A/I	D_Schottky
"D604, "	1	3V3_LED	LED
"F601, "	1	MF-LSMF075X	Polyfuse
"J201, "	1	USB_C_Receptacle	USB_C_Receptacle
"J401, "	1	EXP	Conn_01x04_Female
"J403, "	1	DM3AT-SF-PEJM5	Micro_SD_Card_Det
"J501, "	1	JTAG	Conn_01x06_Male
"JP501, JP502, JP503, "	3	SolderJumper	SolderJumper
"L601, "	1	3.3u	L
"L602, "	1	1.5u	L
"Q201, Q202, Q402, Q403, Q404, ... "	6	PDTC114Y	DTC114Y
"Q203, Q204, "	2	NX7002AK	Q_NMOS_GSD
"Q401, "	1	RZM001P02	Q_PMOS_GSD
"R201, R404, R511, R512, "	4	0R	R
"R202, R203, "	2	5.1k	R
"R204, "	1	22.1k	R
"R205, "	1	47.5k	R
"R206, R302, R303, "	3	1k	R
"R207, R208, "	2	22R	R
"R209, R210, R615, "	3	510R	R
"R301, R408, "	2	4.7k	R

”R304, R305, R619, ”	3	DNP	R
”R306, R505, R611, R613, ”	4	200k	R
”R401, R403, R405, R406, R507, ...”	17	10k	R
”R402, R407, R409, R602, ”	4	330R	R
”R410, R411, ”	2	220R	R
”R501, R502, R503, R504, ”	4	2.2k	R
”R506, R603, R622, R623, ”	4	100k	R
”R605, R609, ”	2	10R	R
”R606, ”	1	31.6k	R
”R608, ”	1	10m	R
”R616, ”	1	4.7R	R
”R617, ”	1	0R BUCK TEST	R
”R618, ”	1	178k	R
”R620, ”	1	86.6k	R
”R621, ”	1	54.9k	R
”RN401, ”	1	56k	R_Pack08
”SW501, ”	1	SW_BOOT	SW_Push
”SW502, ”	1	SW_RST	SW_Push
”SW503, ”	1	SW_LOG_SD_EN	SW_Push
”SW504, ”	1	SW_RST_PROV	SW_Push
”SW505, ”	1	SW_RGB_EN	SW_Push
”SW506, ”	1	SW_SLEEP	SW_Push
”SW601, ”	1	SW_SPDT	SW_SPDT
”TH601, ”	1	NTC	Thermistor_NTC
”U201, ”	1	CP2102N-Axx-xQFN28	CP2102N-Axx-xQFN28
”U301, ”	1	DS18B20	DS18B20
”U302, ”	1	AS7341	AS7341
”U303, ”	1	BH1750FVI-TR	BH1750FVI-TR
”U304, ”	1	TCS34725FN	TCS34725FN
”U305, ”	1	TSL2561	TSL2561
”U306, ”	1	PCA9306GM,125	PCA9306GM,125
”U401, ”	1	PCF2127AT	PCF2127AT
”U402, ”	1	SAM-M8Q	SAM-M8Q
”U501, ”	1	ESP32-WROVER-IE	ESP32-WROVER-IE
”U502, ”	1	SX1503I091TRT	SX1503I091TRT
”U601, ”	1	NCV8164ASN180T1G	NCV8164ASN180T1G
”U602, ”	1	AP62301Z6-7	AP62301Z6-7
”U603, ”	1	MAX77960BEFV12+	MAX77960BEFV12+

A.4 LAITS-Firmware Partition Table

Custom partition table for ESP32 8MB flash memory version with OTA Updates support enabled:

Name	Type	Subtype	Offset	Size	Flags
nvs	data	nvs	0x9000	16k	
otadata	data	ota	0xd000	8k	
phy_init	data	phy	0xf000	4k	
factory	app	factory	0x10000	2M	
ota0	0	ota_0	0x210000	2M	
ota1	0	ota_1	0x410000	2M	